

Computermathematik 2

(Arbeiten mit MATLAB)

Praktikum für BT2 / UV2

Fachbereich MuV, Hochschule Furtwangen University
 Prof. Dr. Stefan von Weber
 Skriptversion 10 / 2011 (Skript 1575)

Wir arbeiten zur Zeit mit dem Release R2008a von MATLAB

Inhaltsverzeichnis

Einführung in MATLAB ®	Seite 2
Aufgabe 1: Erste Schritte mit MATLAB	Seite 5
Aufgabe 2: Reihenentwicklung und einfache Integration	Seite 27
Aufgabe 3: Komplexes Rechnen	Seite 31
Aufgabe 4: Kurvenglättung nach mehreren Methoden, Splines & Zusatzaufgabe	Seite 37
Aufgabe 5: Behälterberechnung mit Daten aus Excel	Seite 42
Aufgabe 6: Funktionstabellierung mit 2D-Plots & Zusatzaufgabe	Seite 45
Aufgabe 7: Elastische Membran (Gleichungssysteme lösen)	Seite 47
Aufgabe 8: Differenzialgleichungssystem gekoppelte Pendel	Seite 53
Aufgabe 9: Integrodifferenzialgleichungssystem mit Nebenbedingungen	Seite 59
Aufgabe 10: Partielle DGL – Die ebene Wellengleichung (Animation)	Seite 63

Allgemeines

Die Veranstaltung „**Computermathematik 2 (Arbeiten mit MATLAB)**“ ist ein von mir betreutes Praktikum ohne begleitende Vorlesung. Das Hauptgewicht liegt auf **MATLAB** und **mathematischen Anwendungen**. Sie sind auf das vorliegende Skript und die Betreuung im Praktikum gestellt. Natürlich können Sie weitere Quellen anzapfen, wie Internet oder Bücher. Ihre wichtigste Hilfe ist jedoch das Team. Arbeiten Sie nicht allein, sondern in einer Gruppe.

Das Skript ist so gegliedert, dass nach einer allgemeinen Einführung in MATLAB zu jedem Aufgabentyp ein Beispiel bearbeitet wird. Ihre Aufgaben sind ähnlich zu den Beispielen zu lösen.

Organisatorisches

Sie bilden Praktikumsgruppen à 2 Personen für die gesamte Zeit des C2-Praktikums. Dafür geht eine Liste um. Falls jemand übrig bleibt, dürfen Sie eine 1-er-Gruppe oder eine 3-er-Gruppe bilden.

Eine 1-er-Gruppe muss 5 von den insgesamt 10 Aufgaben lösen.

Eine 2-er-Gruppe muss 9 von den insgesamt 10 Aufgaben lösen.

Eine 3-er-Gruppe muss alle 10 Aufgaben und die beiden *Zusatzaufgaben für Dreiergruppen* lösen.

Alle Gruppen müssen in ihrer Anzahl immer Aufgabe 1 lösen

Besorgen Sie sich im Dekanat einen Account für die MuV-PC-Hall (Raum A3.14), wenn Sie ihn noch nicht haben. Heben Sie ihn sorgfältig auf. Dieser Account gilt Ihr ganzes Studium lang. Falls das Praktikum in der allgemeinen PC-Hall im D-Bau stattfindet, besorgen Sie sich den Account im Rechenzentrum.

Legen Sie auf Ihrer persönlichen **X-Platte** (samba) einen Ordner **C2** an. Die X-Platte begleitet Sie über das gesamte Studium. Im Ordner C2 legen Sie alle Ergebnisse, Testdaten und Programme ab.

Können Sie eine Aufgabe vorführen, dann wird sie von mir am Rechner angesehen und registriert. Diese **Registrierung ist die Grundlage für den Schein**. Außerdem müssen Sie das **Praktikum mindestens drei Mal besucht** haben. Der Nachweis ist eine Eintragung in die jeweils ausliegende Liste. Ansonsten können Sie auch zu Hause oder in einem anderen Rechnerraum arbeiten. MATLAB ist in der allgemeinen PC-Hall und in der MuV-PC-Hall installiert.

Schaffen Sie es nicht, bis zum Vorlesungsende mir die gelösten Aufgaben zu zeigen, bleiben Ihnen **bis zum Mittwoch der 2-ten Klausurwoche** folgende beiden Möglichkeiten:

- Einen schwarz-weißen Ausdruck in mein Postfach in der FH legen
- Eine Mail an *webers* "at" *fh-furtwangen* senden mit ungezippten PDFs im Anhang.
- Zu einer der angebotenen Nachzüglersessionen zu kommen (Aushang beachten).

1. Einführung in MATLAB

Literatur: Hans Benker: Mathematik mit MATLAB, Springer 2000 und etwa 10.000 Seiten im Internet.

Der Name MATLAB ist aus *Matrix Laboratory* entstanden. MATLAB kann numerische und auch symbolische Aufgaben lösen, d.h. sowohl mit Zahlen rechnen als auch Formeln umwandeln.

MATLAB besteht aus einem allgemeinen Kern und zahlreichen Toolboxen für spezielle Anwendungen. Leider müssen die meisten Toolboxen extra gekauft werden. MATLAB ist ein Produkt der US-Softwarefirma MathWorks.

Es gibt eine verbilligte Studentenversion. Diese gestattet nur kleinere Matrizen und hat nur die 3 Toolboxen Signal Processing, Control System, und Symbolic Mathematics.

Arbeiten mit MATLAB

Im Command Window, dem Arbeitsfenster von MATLAB, erfolgen alle Eingaben, z.B. :

- Text
- Befehle
- Kommandos
- Funktionen
- Ausdrücke

Die meisten Eingaben werden mit der Entertaste (\leftarrow) abgeschlossen.

Eine MATLAB-Sitzung beenden wir mit der Menüleiste oben

→File →Exit (Der Pfeil steht für „linker Mausklick“)

Oder mit dem MATLAB-Kommando im Command Window

>>quit

Die Struktur von MATLAB

MATLAB Benutzeroberfläche	
Kern von MATLAB bestehend aus: Einer Programmiersprache Der Arbeitsumgebung Ein-/Ausgabemöglichkeiten Einige Funktionsbibliotheken Programmschnittstelle, z.B. zu C	Zusatzprogrammbibliotheken (Toolboxen) z.B. Signal Processing Control System (Regelung) Symbolic Mathematics

MATLAB ist **matrixorientiert**, d.h., alle Variablen sind intern als **Felder** (Vektoren, Matrizen) angelegt, auch wenn sie nur eine einzige Zahl enthalten. Die **Architektur** ist offen, d.h., jeder Nutzer kann eigene Algorithmen und Funktionen einbinden. Dazu steht die **Sprache M** zur Verfügung - eine Art abgespecktes C -, aber mit der Möglichkeit, machtvolle Funktionen aus MATLAB aufrufen zu können. Der versierte Anwender kann auch eigene C-Programme oder FORTRAN-Programme einfügen.

Das Command Window (Arbeitsfenster, workspace)

Die Menüleiste des Command Window enthält die Buttons FILE, Edit, Debug, Desktop, Window, Help.

File	Auswahl von Dateien für Eingabe oder Ausgabe
Edit	Textbearbeitung, z.B. Ausschneiden, Kopieren, Wiedereinfügen, Rückgängigmachen
Debug	Unterstützung der Fehlersuche, schrittweise Abarbeitung
Desktop	Benutzeroberfläche konfigurieren
Window	<i>Show Workspace</i> zeigt eine Übersicht des verwendeten Speichers <i>Current Directory</i> zeigt das Dateiverzeichnis
Help	Gibt Hilfestellung zu Kommandos und Funktionen, bietet Demos an

Wir können in zwei unterschiedlichen **Arbeitsmodi** arbeiten.

Interaktiv, d.h. man gibt hinter dem Eingabeprompt `>>` im Arbeitsfenster ein Kommando, eine Variablenzuweisung, oder einen Funktionsaufruf ein und drückt dann die Entertaste (`↵`). MATLAB berechnet die Ergebnisse und zeigt sie auch sofort an.

Nichtinteraktiv, d.h. man startet ein vorgefertigtes Programm, das in der Sprache M abgefasst ist. Das Programm fordert selbständig Daten an, berechnet die Ergebnisse und gibt sie aus.

Schreiben Sie mit dem normalen Texteditor *Editor* ein MATLAB-Programm, z.B.

```
% 3-D Plot analytische Landschaft
[X,Y]=meshgrid(linspace(-3, +3, 50), linspace(-3, +3, 50));
kZ=X+Y.*1i;
Z=abs(1./(1+kZ));
mesh(X,Y,Z)
```

1. Speichern Sie dieses Programm z.B. als *AufgabeX.txt* in Ihren C2-Ordner.
2. Lassen Sie das Editor-Fenster hinter dem MATLAB-Fenster bestehen.
3. Kopieren Sie das Programm in das Command Window von MATLAB. Falls dort schon ein Programm steht, machen Sie `→ Edit → clear Command Window` und `→ Edit → clear Work Space`.
4. Lassen Sie das kopierte Programm laufen (notfalls ENTER-Taste drücken).
5. Wenn Fehler auftreten, korrigieren Sie diese im Editorfenster (Speichern nicht vergessen) und wiederholen die Prozedur ab Punkt 3 bis alles richtig ist.

Benötigt das Programm eine selbst geschriebene Funktion, z.B. *FunktionABC.m*, dann verwalten Sie den Funktionstext zur einfacheren Ausbesserung in einem zweiten Editorfenster. Der Speicherplatz einer Funktion sollte Ihr C2-Ordner auf der X-Platte sein. Die Extension muss bei Funktionen unbedingt *m* sein (nicht *txt*). Den Pfad zu diesem Ordner teilen Sie MATLAB folgendermaßen mit:

`→ File → set Path → Add Folder → X:\C2 auswählen → close → do you wish to... → Nein`

Dateien, Pfadnamen

Zur Arbeit mit Dateien gibt es verschiedene Kommandos.

Kommandoname	Was macht das Kommando
<code>cd</code>	Zeigt das aktuelle Dateiverzeichnis
<code>cd Pfadname</code>	Zeigt die Dateien im verlangten Verzeichnis an z.B. <code>cd C:\MATLAB\TOOLBOX</code>
<code>what</code>	Listet alle M-, MAT-, MEX-Dateien im aktuellen Verzeichnis
<code>what Pfadname</code>	Listet alle M-, MAT-, MEX-Dateien im verlangten Verzeichnis
<code>which Name</code>	Lokalisiert Funktion <i>Name</i> im aktuellen Verzeichnis
<code>dir</code>	Listet alle Unterverzeichnisse zum aktuellen Verzeichnis
<code>dir Pfadname</code>	Listet alle Unterverzeichnisse zum verlangten Verzeichnis

Aufgabe 1: Erste Schritte mit MATLAB (Pflicht für alle)

Aufgabenstellung: Sie nehmen sich alle 22 Beispiele aus den gerahmten Boxen unter Aufgabe 1 vor und spielen sie nach. Dabei machen Sie kleine Änderungen, z.B. Variablennamen ändern, Zahlen ändern, um tiefer in den Sinn der Beispiele einzudringen.

Ergebnisse: Damit ich als Ihr Betreuer eine Kontrolle über Ihr Tun habe, protokollieren Sie alle nummerierten Beispiele mit den beiden Kommandos *diary* und *diary off*. Dabei entstehen auf Ihrer X-Platte im Ordner C2 Textdateien, die man später mit dem *Editor* aus dem *Zubehör* lesen kann. Wie das Protokollieren zu bewerkstelligen ist, sehen Sie am *Beispiel 01*. Sie stellen das Protokoll für *jedes* nummerierte Beispiel her. Ich werde mir beim Kontrollieren der **Aufgabe 1** stichprobenartig einige der entstandenen Textdateien zeigen lassen. Wenn zusätzlich Graphiken und/oder Datenfiles entstehen, dann speichern Sie Graphiken unter dem Namen *Beispiel??.tif* und Daten unter *Matrix??.txt* ab. Die beiden Fragezeichen stehen für die Beispielnummer.

1.1 Interaktives Arbeiten im Command Window (Arbeitsfenster)

Kommandos oder Rechenanweisungen werden mit der Entertaste (\leftarrow) abgeschlossen. Möchte man jedoch mehrere Eingaben machen und dann erst rechnen, dann trennt man die Anweisungen mit Semikolon (;). Ein Zeilenumbruch nach einem Semikolon löst keine Berechnung aus. Erst eine Anweisung ohne Semikolon und anschließend die Entertaste (\leftarrow) löst dann die Berechnung aus.

Hat ein Ausdruck keine Zuweisung zu einer benannten Variablen, dann wird automatisch die allgemeine Variable **ans** verwendet (Vom Englischen *answer*). Text, der mit dem Prozentzeichen (%) beginnt, ist Kommentar. Im Beispiel 1 berechnen Sie zuerst ein Integral, dann eine Formel mit a=2 und b=3.

$$\text{Flaeche} = \int_0^{1.2} \cos(x) dx$$

$$\left(\frac{1}{a} + \sqrt{5} \right) / \left(b^3 + \sqrt[5]{7} \right) \quad \text{mathematisch ist } \sqrt[5]{7} = 7^{(1/5)}$$

Beispiel 01:

```
>> diary X:\C2\Bsp01.txt  $\leftarrow$            % Sie starten das Protokoll zu Beispiel 01
>> Flaeche=quad('cos', 0, 1.2)  $\leftarrow$       % („quad“ steht für Quadrature (Integration))
    Flaeche = 0.9320

>> a=2; b=3;  $\leftarrow$ 
>> (1/a + sqrt(5)) / ( b^3 + 7^(1/5) )  $\leftarrow$  % Das Ergebnis wird nach ans gespeichert
    ans = 0.0961

>> diary off  $\leftarrow$                        % Sie beenden das Protokoll
```

Wir können die Resultate und Variablenbelegungen einer Sitzung sichern (wenn wir wollen) z.B. am 18. März 2011. Dazu gehen wir oben in die Menüleiste:

→FILE →Save Workspace as ... Sitzung18032011.mat

Dateien der Art *.mat sind Binärdateien, können also nicht als Text gelesen werden. Starten wir eine neue Sitzung, dann können wir jedoch die Variablenbezeichnungen mit ihren Zahlenwerten wieder einlesen

→FILE →Import Data → Sitzung18032011.mat

Das Eingelesene bleibt vorerst unsichtbar. Schreiben wir jedoch einen in der Sitzung verwendeten Variablennamen in das Command Window und geben Enter, dann erscheint der Wert der Variablen, z.B.

```
>> ans ←↵
ans = 0.0961
```

Eine zweite Art der Sicherung haben Sie schon kennengelernt. Sie erfolgt mit den Kommandos **diary Pfadname** und **diary off** (diary = Tagebuch). Das nachfolgende Beispiel zeigt die Verwendung mit einem Speicherstick z.B. im Laufwerk F.

Die Sitzung im Command Window	Beschauen wir uns die Textdatei auf dem Stick, dann finden wir folgenden Text:
<pre>>> diary F:\Sitzung18032011.txt ←↵ >> a=2; b=3; ←↵ >> K=sqrt(a^2+b^2) ←↵ K = 3.6056 >> diary off ←↵</pre>	<pre>a=2; b=3; K=sqrt(a^2+b^2) K = 3.6056 diary off</pre>

Diesen Text können Sie in ein anderes Programm einfügen, z.B. in eine M-Datei, oder Sie können den Text in ein Word-Dokument als Beispiel einfügen.

Beispiel für save und load: Die Werte werden binär gespeichert

<p>Sitzung z.B. am Montag: endet mit einem save</p>	<pre>>> a=pi ←↵ >> a = 3.1416 >> b=exp(1) ←↵ >> b = 2.7183 >> save F:\Sitzung25102011.mat ←↵</pre>
<p>Sitzung am Dienstag startet mit load Die Werte vom Montag sind wieder verfügbar</p>	<pre>>> load f:\Sitzung25102011.mat ←↵ >> a ←↵ a = 3.1416 >> b ←↵ b = 2.7183</pre>

1.2 Zeichenkettenfunktionen, Textausgabe

Das sind Funktionen, die Zeichenketten (*Textzeilen*) herstellen aus Textbausteinen und berechneten Zahlen, z.B. zur Beschriftung von Graphiken. Die Funktion **sprintf** ist eine dieser Funktionen. Sie stellt eine Textzeile (string) her und setzt in den Text Zahlen ein.

Beispiel 02:

```
>> r=0.5; h=2; V=pi*r^2*h; %Volumen Zylinder ←↵
>> sprintf('Kreiszyylinder mit \n Radius=%g und Höhe %g \n hat das Volumen %g', r,h,V)←↵
ans= Kreiszyylinder mit
      Radius=0.5 und Höhe 2
      hat das Volumen 1.5708
```

Die Zeichenfolge „\n“ bewirkt im erzeugten Text den Übergang auf eine neue Zeile. Die Zeichenfolge „%g“ heißt Formatspezifikation und fungiert als Platzhalter für eine einzusetzende Zahl. Die Zahl, die eingesetzt wird, steht in der Liste am Ende der sprintf-Anweisung (hier im Beispiel die Liste der drei Variablen *r*, *h*, und *V*.) Die Einsetzung erfolgt von links nach rechts, d.h. *r* landet auf dem ersten Platzhalter, *h* auf dem zweiten usw.

Will man nur Text ausgeben, z.B. eine Fehlermeldung aus einem automatisch ablaufenden MATLAB-Programm heraus, dann ist die Funktion `disp()` einfacher zu handhaben. Man kann aber auch Zahlen ausgeben, aber nicht beides, d.h. Text und berechnete Zahl zusammen. Man muss die `disp`-Funktion getrennt für den Text und die Zahl aufrufen.

Beispiel 03:

```
>> disp('Fehler 007: Koeffizient b ist negativ'); % reine Textausgabe ←↵
>> r=2.9 ←↵
>> V=(4*pi/3)*r^3; ←↵
>> disp('Das Volumen ist '); disp( V ); % erst Text, dann die Zahl extra ←↵
```

1.3 Längere Berechnungen

Man kann die einzelnen Anweisungen mit Semikolon abschließen. Dann werden sie erst ausgeführt, wenn eine Anweisung ohne Semikolon, aber mit der Entertaste folgt (↵). Im Beispiel sollen die beiden Wurzeln *x1* und *x2* der quadratischen Gleichung $y = x^2 + a x + b$ berechnet werden.

Beispiel 04:

```
>> % Berechnung der beiden Wurzeln einer quadratischen Gleichung ←↵
>> a=4; b=5; % Die Wertzuweisung an die Koeffizienten ←↵
>> x1=-a/2+sqrt(a^2/4-b) ←↵
x1 = -2.0000 + 1.0000i
>> x2=-a/2-sqrt(a^2/4-b) ←↵
```

```

x2 = -2.0000 - 1.0000i

>> % Wir berechnen das Produkt der beiden Komplexen Zahlen
>> produkt=x1*x2
    produkt = 5

>>% Ein anderes Beispiel
>>SQ=3177.234; mittel=19.321; n=5;
    sigma=sqrt( (SQ-n*mittel^2)/(n-1)) ←
    sigma = 18.1020

```

1.4 Hilfe

Hilfe kann man in MATLAB auf verschiedene Weise erhalten:

Menü → Help → Examples und Demos

oder als Beispiel der Hilfe zur Funktion „quad“ (Quadratur) mit dem help-Kommando: Sie bekommen Hinweise zur Anwendung, zur Bedeutung der anzugebenden Argumente und Anwendungsbeispiele.

Beispiel 05:

```

>> help quad ←
QUAD Numerically evaluate integral, adaptive Simpson quadrature.
Q = QUAD(FUN,A,B) tries to approximate the integral of scalar-valued
function FUN from A to B to within an error of 1.e-6 using recursive
adaptive Simpson quadrature. FUN is a function handle. The function
Y=FUN(X) should accept a vector argument X and return a vector result
Y, the integrand evaluated at each element of X.
.....
function_handle">function_handle</a>.

Reference page in Help browser
<a href="matlab:doc quad">doc quad</a>

```

Oder mit dem type-Kommando: Sie bekommen den gesamten Quelltext der Funktion „quad“. Dieser ist jedoch mehr für Profis interessant, die solche Funktionen programmieren.

Beispiel 06:

```

>> type quad ←

function [Q,fcnt] = quad(funfcn,a,b,tol,trace,varargin)
%QUAD Numerically evaluate integral, adaptive Simpson quadrature.
% Q = QUAD(FUN,A,B) tries to approximate the integral of scalar-valued
% function FUN from A to B to within an error of 1.e-6 using recursive
.....
.....

```

1.5 Symbolisches Rechnen

MATLAB unterscheidet zwischen exakter (symbolischer oder formelmäßiger) Lösung und numerischer Lösung. Wir konzentrieren uns später ausschließlich auf numerische Lösungen.

Im Beispiel wird zuerst die exakte symbolische Lösung des Integrals $\int_0^{\pi} \sin(x) dx$ bestimmt.

Das Integral des Sinus ist $-\cos(x)$ und liefert für das Intervall $[0, \pi]$ nach einer Rechenzeit von ca. zwei Sekunden exakt den Wert 2. Weiteres Beispiele für symbolisches Rechnen ist hier eine Bruchrechnung.

Beispiel 07:

```
>> syms x; int(sin(x), 0, pi) ←⌋      bzw. x = sym('x'); int(sin(x), 0, pi) ←⌋
ans = 2

>> sym(1/3+1/7) ←⌋      (oder auch >> sym(1/3)+sym(1/7) ←⌋)
ans = 10/21

>> double(ans) ←⌋      ( Ausgabe des symbolischen Resultats als Dezimalzahl )
ans = 0.4762
```

Die numerische Berechnung erfordert hingegen keine merkliche Rechenzeit:

Beispiel 08:

```
>> quad('sin', 0, pi) ←⌋
ans = 2.0000

% mit der Funktion vpa kann man die Zahl der ausgegebenen Dezimalstellen steuern.
% Die Berechnung wird jedoch symbolisch durchgeführt und ist deshalb für längere
% und kompliziertere Rechnungen nicht zu empfehlen

>> vpa(1/3+1/7, 12) ←⌋
ans = .476190476190      % ist intern als Formel gespeichert (Sym)

>>help vpa ←⌋

VPA Variable precision arithmetic.
R = VPA(S) numerically evaluates each element of the double matrix
S using variable precision floating point arithmetic with D decimal
digit accuracy, where D is the current setting of DIGITS.
The resulting R is a SYM. ....
.....
```

Symbolische Differenzieren und Integrieren in MATLAB am Beispiel der Funktion $y=x*\sin(x)$

Beispiel 09:

Symbolisches Differenzieren	Symbolisches Integrieren
<pre>% Zuerst die erste Ableitung >> syms x; bzw. x=sym('x'); ←┘ >> diff(x*sin(x), x, 1) ←┘ ans = sin(x)+x*cos(x) % Jetzt die zweite Ableitung >> diff(x*sin(x), x, 2) ←┘ ans = 2*cos(x)-x*sin(x)</pre>	<pre>% Wir integrieren die zweite Ableitung >> int(ans, x) ←┘ ans = sin(x)+x*cos(x) % Wir integrieren nochmals >> int(ans,x) ←┘ ans = x*sin(x)</pre>

1.6 Umgang mit Variablen

Jede Variable kann immer wieder neu belegt werden, so auch die Variable „ans“. Aber auch ein berechneter Wert kann mehrmals für andere Berechnungen herangezogen werden.

Beispiel 10:

Änderung der Belegung einer Variablen	Mehrfache Nutzung eines Wertes
<pre>>> vpa(1/3+1/7, 12) ←┘ ans = .476190476190 >> 1/3+1/7 ←┘ ans = 0.4762</pre>	<pre>>> V=ans^2 ←┘ V = 0.2268 >> U=ans*2+3 ←┘ U = 3.9524</pre>

Die Stellenzahl der ausgegebenen Resultate kann mit dem Kommando „format“ grob gesteuert werden:

>> format short	veranlasst die Ausgabe von Festkommazahlen mit 5 Ziffern
>> format long	veranlasst die Ausgabe von Festkommazahlen mit 15 Ziffern
>> format short e	veranlasst die Ausgabe von Exponentialzahlen mit 5 Ziffern
>> format long e	veranlasst die Ausgabe von Exponentialzahlen mit 15 Ziffern
>> format short g	wählt automatisch Festkomma oder Exponentialdarstellung
>> format long g	wählt automatisch Festkomma oder Exponentialdarstellung

Beispiel 11:

```
>> format long;
>> 100/7 ←␣

ans = 14.285714285714286

>> z=1+2i; abs(z) ←␣

ans = 2.236067977499790

>> format short; angle(z) ←␣

ans = 1.1071
```

1.7 Komplexe Zahlen

Zahlen liegen zwischen $\pm 10^{-308}$ und $\pm 10^{+308}$. MATLAB kennt die Werte von π und e . Ganze Zahlen bestehen nur aus Ziffern.

Dezimalzahlen (Gleitkommazahlen) enthalten einen Punkt und/oder einen Exponenten.

Komplexe Zahlen enthalten die imaginäre Einheit „i“ bzw. „j“, z.B. $z=a+b \cdot i$ oder $z=a+b \cdot j$.

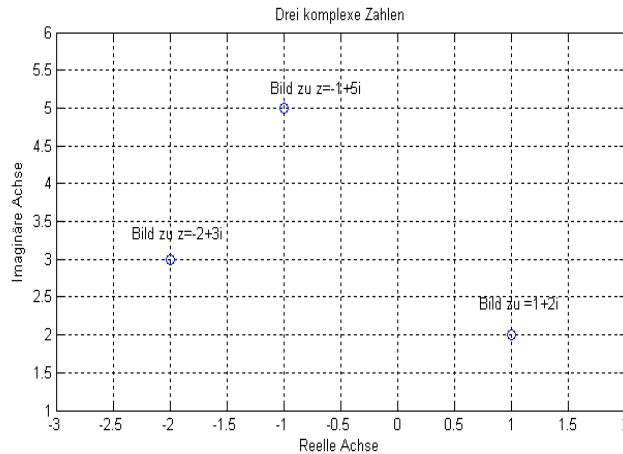
Die komplexe Funktion $\text{abs}(z)$ liefert den Betrag $r = \sqrt{a^2 + b^2}$
 $\text{angle}(z)$ liefert den Winkel $\varphi = \arctan(b/a)$ im Bogenmaß
 $\text{real}(z)$ liefert den Realteil (hier a) als reelle Zahl
 $\text{imag}(z)$ liefert den Imaginärteil (hier b) als reelle Zahl

Beispiel 12:

<pre>>> z= 1 + 2i; abs(z) ←␣ ans = 2.2361 >> angle(z) ←␣ ans = 1.1071</pre>	<p>Belege z mit einer komplexen Zahl und bilde den Betrag Berechne den Winkel zu z in Bogenmaß</p>
<pre>>> z1=1+2i; z2=-2+3i; >> z= [z1, z2, z1+z2]; >> plot(z, 'o') ←␣</pre>	<p>Belege z1 und z2 mit komplexen Werten. Bilde einen Vektor z mit 3 Elementen. Plote die 3 Elemente als kleine Kreise auf der Gaußschen Zahlenebene.</p>

Den Plot bearbeiten wir interaktiv und speichern ihn als Bild ab. Die Gaußsche Zahlenebene hat die reelle Achse x und die imaginäre Achse iy . Eine komplexe Zahl ist ein Punkt auf der Gaußschen Zahlenebene. Die drei komplexen Zahlen aus Vektor z werden demnach als 3 Punkte dargestellt (hier durch kleine Kreise gemäß Plotsymbol 'o').

Das Gitter, die Achsenlabels, der Titel und die Punktbeschriftungen wurden hier „per Hand“ mit dem Insertmenü aus der Menüleiste der Graphik eingefügt und das Bild dann als TIF-Datei exportiert.



1.8 Variablen

MATLAB unterscheidet einfache und indizierte Variable.

Einfache Variable sind z.B. a A Abholgewicht_x x17 Karo_As
 Indizierte Variable sind z.B. x(3) x(k) Kostenmatrix(4,7)

Nicht erlaubt als Variablennamen sind die Namen von vordefinierten Funktionen, Konstanten oder von Kommandos. Namen beginnen immer mit einem Buchstaben und dürfen keine Blanks enthalten. Nehmen Sie stattdessen das Unterstreichungszeichen wie in Karo_As. Groß-Kleinschreibung ist wichtig. Die Variablen a und A sind unterschiedliche Variablen. Variablen können ihre Werte im Verlauf einer Rechnung ändern, Konstanten nicht.

Das erste Auftreten einer Variablen im Programm definiert sie. Vermeiden Sie es, dass Variablen innerhalb von Schleifen oder Verzweigungen definiert werden. Ausnahmen sind die Laufindizes von Schleifen, z. B. das x in der Schleife for x=1:10;; end; Benötigen Sie innerhalb von Schleifen oder Verzweigungen neue Variable, dann nennen Sie sie vor dem Beginn der Schleife, z.B. in einer (manchmal sinnlosen) Zuweisung, z.B.:

```
A=0; z=0; MatrixS=0;
```

```
>> clear A ←┐      löscht Variable A (Name und Wert sind weg)
>> clear all ←┐    löscht alle Variablen (Namen und Werte sind alle weg)
```

Numerische Variablen enthalten Zahlen oder aber Texte in Form von Zeichenketten. Symbolische Variablen enthalten Formeln.

```
>> who ←┐      bzw. >> whos ←┐      liefert eine Übersicht über alle Variablen.
```

Beispiel 13:

```
>> a=5; b=3;
>> who ←┐
```


```
Your variables are:  a  b
```

```
>> whos ←┐
```

```
Name      Size      Bytes Class      Attributes
```

Wir belegen zwei Variablen a und b. Wir wollen uns über unsere Variablen informieren (wie viele, welche Namen).
Wir wollen mehr über unsere Variablen wissen:

a	1x1	8	double	
b	1x1	8	double	

Den gleichen Effekt liefert ein Klick auf den Button  in der Menüleiste.

Vordefinierte Konstanten und Variable

π	als	pi	Kreiszahl
e	als	$exp(1)$	Eulersche Zahl
i	als	i oder j	Imaginäre Einheit
∞	als	Inf oder inf	Unendlich (engl. infinite)
ε	als	eps	Kleinste Zahl, die zu 1 addiert eine Zahl >1 ergibt (Genauigkeit des PC)

realmin ist die kleinste im PC darstellbare Gleitkommazahl
 realmax ist die größte im PC darstellbare Gleitkommazahl
 NaN Not a Number (der Wert der Zahl ist unbestimmt, z.B. nach 0/0)

Einige Beispiele sollen den Umgang mit diesen Größen vertiefen.

Beispiel 14:

<pre>>>a=pi ← a = 3.1416 >>e=exp(1) ← e = 2.7183 >>k=inf ← k = Inf >>z=eps ← z = 2.2204e-016 (2,2204·10⁻¹⁶) >>a=realmin ← a = 2.2251e-308 (2,2251·10⁻³⁰⁸) >>a=realmax ← a = 1.7977e+308 (1,7977·10⁺³⁰⁸) >>k=0/0 ← (Null durch Null) k = NaN (is Not a Number)</pre>	<pre>>>a=inf+inf ← (∞ + ∞ ergibt ∞) a = Inf >>a=exp(1000) ← (e¹⁰⁰⁰ gibt ∞) a = Inf >>a=exp(700) ← (e⁷⁰⁰ klappt) a = 1.0142e+304 (1,0142·10⁺³⁰⁴) >>a=i ← (imaginäre Einheit) a = 0 + 1.0000i (komplexe Zahl) >>a=1/0 ← (Division durch 0) a = Inf (ergibt ∞) >>a=inf-inf ← (∞ - ∞ ergibt) a = NaN (keine Zahl)</pre>
--	---

1.9 Zeichenkettenfunktionen

Diese Funktionen benötigt man z.B. bei der automatischen Umwandlung von Pfadnamen, oder bei der Zusammenstückelung von Textbausteinen für die Beschriftung von Graphiken.

`v=abs(z)` liefert die ASCII-Zahlencodes der Zeichen von Zeichenkette `z` als Vektor `v`. ASCII (American Standard Code for Information Interchange). Der Code kann 255 verschiedene Zeichen codieren, jeweils ein Zeichen ist ein Byte bzw. eine 8-Bit-Zahl zwischen 0 und 255.

`z=char(v)` macht aus dem Zahlenvektor `v` eine Zeichenkette `z`. Bedingung ist jedoch, dass die Zahlen im Vektor nur Werte zwischen 0 und 255 haben (ASCII-Codes).

`d=double(z)` arbeitet wie `abs(z)`, nur dass double-Zahlen als Vektorelemente von Vektor `d` erzeugt werden.

`b=isstr(vz)` liefert `b=1`, wenn Argument `vz` eine Zeichenkette ist, sonst `b=0`.

`b=ischar(vz)` liefert `b=1`, wenn Argument `vz` eine Zeichenkette ist, sonst `b=0`.

`b=strcmp(z1,z2)` liefert `b=1`, wenn die beiden Zeichenketten `z1` und `z2` identisch sind, sonst wird `b=0`.

`n=length(z)` liefert die Anzahl der Zeichen in der Zeichenkette `z` einschließlich der Blanks.

`z=num2str(x)` wandelt die Gleitkommazahl `x` in eine Zeichenkette um.

`z=int2str(k)` wandelt die ganze Zahl (Integerzahl) `k` in eine Zeichenkette `z` um.

`z=mat2str(M)` wandelt Matrix `M` in eine Zeichenkette um.

`x=str2num(z)` wandelt die Zeichenkette `z` in eine Zahl `x` um, falls das möglich ist, d.h. die Zeichenkette `z` sollte schon die Gestalt eine Zahl haben, z.B. 17 oder 3.14 oder 3.27e-05

`z=sprintf('Format',Liste)` wandelt die Zahlen aus der Liste in Zeichenketten um und fügt diese dann statt der Platzhalter in das Format ein. Es entsteht eine lange Zeichenkette `z`, die mehrere Texte und Zahlen enthalten kann (siehe Beispiel 02).

`z1=lower(z)` wandelt alle Großbuchstaben (und nur diese) von Zeichenkette `z` in Kleinbuchstaben um. Alle anderen Zeichen werden unverändert übernommen.

`z1=upper(z)` wandelt alle Kleinbuchstaben (und nur diese) von Zeichenkette `z` in Großbuchstaben um. Alle anderen Zeichen werden unverändert übernommen.

1.10 Datentypen – Felder

Vektoren und Matrizen heißen Felder. Sie beanspruchen entsprechend ihrer Größe viel Speicherplatz. Auf die Vektorelemente bzw. Matrixelemente wird mit Indizes zugegriffen. Statt x_7 schreibt man in MATLAB jedoch `x(7)`, oder statt A_{ij} schreibt man `A(i,j)`.

Die Zahlenwerte der Vektorelemente kann man eintippen oder aus einer Datei übernehmen oder berechnen. Die folgenden Beispiele zeigen den Umgang mit Vektoren und Matrizen.

Beispiel 15:

```

>>[3 -2 5 7 9] ←↵
ans = 3 -2 5 7 9

>>a=[3 -1 9; 4 2 0; -9 8 3; 2 2 1] ←↵
a = 3 -1 9
    4 2 0
   -9 8 3
    2 2 1

>>a=3; b=7; x=a:b ←↵
x = 3 4 5 6 7

>>x=2.5:0.2:7.3 ←↵

x = Columns 1 through 9

2.5000 2.7000 2.9000 3.1000
..... 3.7000 3.9000 4.1000

Columns 10 through 18

4.3000 4.5000 4.7000 4.9000
.....
.....
6.9000 7.1000 7.3000

>>A=[[3 -1 9];[4 2 0];[-9 8 3];[2 2 1]] ←↵

A = 3 -1 9
    4 2 0
   -9 8 3
    2 2 1

>>A1=[1 2 3 4;5 6 7 8];
>>A2=[3 2 5 1;7 5 3 6];
>>A3=[2 1 3 7;9 7 8 4];
>>A=cat(3,A1,A2,A3) ←↵

A(:,:,1) = 1 2 3 4
           5 6 7 8
A(:,:,2) = 3 2 5 1
           7 5 3 6
A(:,:,3) = 2 1 3 7
           9 7 8 4

>>u=A(1,2,3) ←↵

```

Belegen eines unbenannten Vektors mit fünf Elementen.

Belegen der Matrix *a* mit 4 Zeilen zu je 3 Elementen. Zeilen werden durch Semikolons getrennt. Die Elemente in einer Zeile werden durch Kommas oder Blanks getrennt.

Belegen des Vektors *x* mit Elementen, wobei die Zahlenwerte durch eine implizite (oder implizierte) Schleife „von : bis“ vorgegeben werden. Schrittweite ist 1. Es sind jedoch beliebige Schrittweiten möglich, wie das Beispiel links zeigt („von : Schrittweite : bis“)

Wie man sieht, kann man leicht 1000 Seiten Bildschirm füllen, wenn man übertreibt.

Vektoren können als Elemente auch wieder Vektoren enthalten. Es entsteht ebenfalls eine Matrix, die intern aber anders organisiert ist, als eine Matrix aus Zeilen gleicher Länge. (Mehr etwas für Profis.)

3-dimensionale Matrizen $\{A_{ijk}\}$ kann man sich zusammengesetzt denken als zweidimensionale Matrizen, die man hintereinander stellt wie Karteikarten: Karte 1, dahinter Karte 2, usw.

MATLAB zeigt jede Karte einzeln. Die seltsame Indizierung $A(:, :, 1)$ mit den beiden Doppelpunkten und der 1 bedeuten, dass die ersten beiden Indizes laufen, während der dritte Index den Wert 1 hat (1. Karteikarte). Dann kommt die zweite usw.

Wir wollen ein bestimmtes Matricelement

<pre> u = 1 >>u=A(2,2,2) ← u = 5 >>i=1; j=4; k=3; u=A(i,j,k) ← u = 7 >>u=A(i+1, j-2, k-i) ← u = 5 </pre>	<p>auf die Variable u umspeichern. Zeile 1, dort das zweite Element, alles auf der dritten Karte - dort steht tatsächlich der Wert 1. Zeile 2, zweites Element, alles auf der zweiten Karte. Wir finden den Wert 5.</p> <p>Indizes dürfen auch ganzzahlige Variable sein oder ganzzahlige Ausdrücke.</p> <p>Die 3 Indizes haben die Werte $i+1=2$, $j-2=2$, $k-i=2$. Wir finden Element $A(2,2,2) = 5$.</p>
---	---

Für Felder sind folgende Rechenoperationen erlaubt:

+ - .* ./ .^ Elementweises Addieren, Subtrahieren, Multiplizieren, Dividieren und Potenzieren. Die beteiligten Felder müssen jedoch gleiche Gestalt haben, d.h., bei Vektoren gleiche Elementezahl, bei Matrizen gleiche Zeilen- und Spaltenzahl.

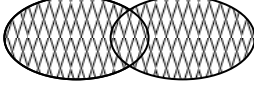

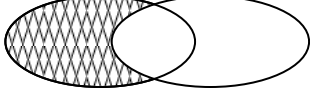
Beispiel 16: Arbeiten mit Feldern

<pre> >>a=[2 3; 4 -2] ← a = 2 3 4 -2 >>b=[1 9; 0 4] ← b = 1 9 0 4 >>g=a-b ← g = 1 -6 4 -6 >>h=a+b ← h = 3 12 4 2 >>k=a.*b ← k = 2 27 0 -8 >>d=b./a ← d = 0.5000 3.0000 0 -2.0000 >>C=a*b ← </pre>	<p>Wir belegen eine Matrix a.</p> <p>Dann Matrix b.</p> <p>Wir subtrahieren a-b.</p> $a-b = \begin{bmatrix} 2 & 3 \\ 4 & -2 \end{bmatrix} - \begin{bmatrix} 1 & 9 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & -6 \\ 4 & -6 \end{bmatrix}$ $a+b = \begin{bmatrix} 2 & 3 \\ 4 & -2 \end{bmatrix} + \begin{bmatrix} 1 & 9 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 4 & 2 \end{bmatrix}$ <p>Jedes Element von Matrix a wird mit dem entsprechenden Element von Matrix b multipliziert.</p> $\begin{bmatrix} 2 \cdot 1 & 3 \cdot 9 \\ 4 \cdot 0 & (-2) \cdot 4 \end{bmatrix} = \begin{bmatrix} 2 & 27 \\ 0 & -8 \end{bmatrix}$ <p>Auch elementweises Dividieren ist möglich. (Vorsicht bei Nullen!!). Die Ergebnisse können jedoch gebrochene Zahlen sein.</p> <p>Die übliche Matrixmultiplikation folgt den</p>
--	--

<pre> C = 2 30 4 28 >>Inverse=a^(-1) ← Inverse = 0.1250 0.1875 0.2500 -0.1250 >>E=Inverse*a ← E = 1 0 0 1 x=[1:5]; y=[-2: 2]; ← Centers=['Ulm '; 'Wien'; 'Bonn'; 'Lund'; 'Graz']; ← plot(x,y,'o'); ← axis([0 8 -3 4]); ← text(x+0.2, y+0.2, Centers); ← [x,y]=meshgrid([1 2 3 4],[0.1 0.2 0.3]); ← x ← x = 1 2 3 4 1 2 3 4 1 2 3 4 Y ← y = 0.1000 0.1000 0.1000 0.1000 0.2000 0.2000 0.2000 0.2000 0.3000 0.3000 0.3000 0.3000 z=x; ← z ← z = 1 2 3 4 1 2 3 4 1 2 3 4 z=randn(size(z)); z ← z = -0.4326 0.2877 1.1892 0.1746 -1.6656 -1.1465 -0.0376 -0.1867 0.1253 1.1909 0.3273 0.7258 z(5) ← ans = -1.1465 >> x=zeros(5,7); ← >> x(3:5,4:6)=1; ← >> x ← </pre>	<p>Verkettungsregeln. Man kann eine (m,n)-Matrix a mit einer (n,p)-Matrix b multiplizieren und erhält eine (m,p)-Matrix C, d.h., eine Matrix mit m Zeilen und p Spalten.</p> <p>Die inverse Matrix lässt sich nur von einer (m,m)-Matrix berechnen, d.h. nur von einer quadratischen Matrix.</p> <p>Multipliziert man eine Matrix mit ihrer Inversen, erhält man die Einheitsmatrix.</p> <p>Wir belegen die Vektoren x und y für ein Beispiel mit einem Cell Array of strings, d.h. Zeichenketten. Die Zeichenketten müssen unbedingt gleichlang sein (notfalls mit Blanks auffüllen). Plote kleine „o“ auf die Koordinaten. Mache die Achsen länger als nötig. Beschriftet die Punkte mit obigen Zeichenketten. Erzeuge Gitterpunktkoordinaten für ein Weiteres Beispiel. Zeige die x-Koordinaten an.</p> <p>Zeige die y-Koordinaten an.</p> <p>Umspeichern einer Matrix. Anzeigen.</p> <p>Erzeuge Matrix mit Dimension von z mit normalverteilten Zufallszahlen.</p> <p>Zeige 5. Element von Matrix z an. MATLAB nummeriert die Elemente spaltenweise ab 1.</p> <p>Herstellung einer (5,7)-Nullmatrix. Belegung eines Teils mit 1-en.</p>
---	--

1.11 Mengen

Endliche Mengen werden als Vektoren behandelt. Die Elemente der Mengen sind in MATLAB ganze Zahlen oder Zeichenketten.

Vereinigungsmenge: Doppelte Elemente werden nur einmal übernommen	 $C = A \cap B$	<pre>>>A=[1:10] ← A = 1 2 3 4 5 6 7 8 9 10 >>B=[7:12] ← B = 7 8 9 10 11 12 >>C=union(A,B) ← C = 1 2 3 4 5 6 7 8 9 10 11 12</pre>
Schnittmenge: Nur Elemente, die in beiden Mengen sind, werden übernommen	 $D = A \cup B$	<pre>>>D=intersect(A,B) ← D = 7 8 9 10</pre>
Mengendifferenz: Nur Elemente von B, die auch in A sind, werden aus A entfernt	 $E = A - B$	<pre>>>A=[1:7]; ← >>B=[3 4 6 8 9]; ← >>setdiff(A, B) ← ans = 1 2 5 7</pre>

1.12 Ein-/Ausgabe von Daten

Daten liegen zumeist als Tabellen vor aus Zeilen und Spalten. Bei der Eingabe werden Blanks und Zeilenumbrüche als Trennzeichen zwischen den Zahlen akzeptiert (Blanks für Zeilenelemente, d.h. Zahlen aus einer Zeile, Zeilenumbrüche für den Übergang auf die nächste Tabellenzeile).

Mit dem **load-Kommando** sind ASCII-Dateien lesbar. Diese können z.B. aus Excel exportiert werden. (ASCII-Code: American Standard Code of Information Interchange). Mit dem **save-Kommando** geben wir Daten als Textdateien im ASCII-Code aus.

Will man Zahlen und Text vermischen, sollte man formatiert schreiben (fprintf) oder formatiert lesen (fscanf). Bei Anwendung dieser Funktionen muss die Datei zuvor mit der **fopen-Anweisung** geöffnet werden. Die allgemeine Form ist:

Handle = fopen(Pfadname, Zugriffsrichtung)

Das Handle dient als Kurzzeichen der geöffneten Datei im weiteren Programmverlauf. Der *Pfadname* kann eine Zeichenkette sein oder eine Zeichenkettenvariable. Die Zugriffsrichtung gibt an, ob gelesen oder geschrieben werden soll. Beides gleichzeitig für eine einzelne Datei ist nicht möglich, man kann jedoch eine Datei lesen und auf eine andere Datei schreiben.

Die *Zugriffsrichtung* ist eine kurze Zeichenkette: Zeichenkette 'r' bzw. 'rt' öffnet die Datei für Lesen, Zeichenkette 'wt' öffnet die Datei für das Schreiben von Text.

Vorsicht: Eine schon vorhandene Datei gleichen Namens wird erbarmungslos überschrieben !!!

Die Anweisung `aus = fopen('f:\MatrixA.txt', 'wt')` öffnet auf Laufwerk *F:* eine neue Datei *MatrixA.txt* für die Ausgabe von Text. Das Dateikurzzeichen (Handle) ist *aus*.

Man holt sich den Pfadnamen und den Dateinamen für die Eingabe einer Textdatei am besten über den üblichen **Dateidialog** von Windows mittels der Funktion `uigetfile(...)`, setzt den vollständigen Pfad aus Pfadname und Dateiname zusammen (Funktion `strcat(...)`) und öffnet anschließend die Datei:

```
[Dateiname, Pfadname, Filternummer]=uigetfile( '*.txt' );
Pfad=strcat(Pfadname, Dateiname);
ein=fopen(Pfad, 'rt');
```

Haben wir in `uigetfile` mehrere Extensions zur Auswahl angegeben, dann gibt die Filternummer an, welche Extension die Datei hat.

Dieselbe Mimik gilt für die Ausgabe einer Textdatei. Die Funktion `strcat` wurde hier zur Abwechslung gleich in den `fopen`-Aufruf integriert:

```
[Dateiname, Pfadname, Filternummer]=uigetfile('*.txt');
aus=fopen( strcat(Pfadname,Dateiname), 'wt');
```

Die **formatierte Ein-/Ausgabe** von Daten:

Mit **fprintf** schreiben wir Text und Zahlen in Zeilenform. Die allgemeine Form ist:

```
fprintf( Handle, Format, Ausgabeliste)
```

Das *Format* hat die Gestalt einer Zeichenkette. Das Format enthält Texte, Zeilenvorschübe und gibt an, an welcher Position Texte bzw. Zahlen einzusetzen sind (Platzhalter). Die Ausgabeliste sagt, welche Zahlen auszugeben sind. Das Handle muss zu einer Ausgabedatei gehören. Zum Beispiel schreibt die Anweisung

```
fprintf(aus, '\n %5i Element %10.5f ', k, A(k));
```

auf Datei *aus* zuerst einen Zeilenvorschub (`\n`), dann kommen 2 Blanks, dann wird an der Stelle `%5i` der Wert von *k* aus der Ausgabeliste als ganze Zahl eingefügt mit 5 Druckstellen (*i* wie *integer*), dann wieder 2 Blanks, dann das Wort *Element*, dann wieder 2 Blanks, dann an der Stelle `%10.5f` wird das Vektorelements *A(k)* aus der Ausgabeliste eingefügt mit 10 Druckstellen insgesamt, davon 5 Dezimalstellen. Das *f* steht für *float*, dem englischen Wort für Gleitkommazahlen.

Geöffnete Dateien müssen wir zum Ende der Bearbeitung unbedingt wieder schließen. Das machen wir mit der Anweisung `fclose(Handle)`, z.B. `fclose(aus)`

Mit `fscanf` geschriebene Daten können wir mit der **fscanf-Anweisung** lesen. Die allgemeine Form ist:

```
Zielvariable = fscanf( Handle, Format, Werteanzahl)
```

Das Handle muss das einer Eingabedatei sein. Das Format gibt Zeilenvorschübe in den Daten an und Formatspezifikationen für die einzulesenden Werte. Die Werteanzahl gibt an, wie viel

Werte insgesamt zu lesen und auf die Zielvariable zu speichern sind. Zum Beispiel liest die Anweisung

```
B=fscanf(ein, '\n%g', 10);
```

zehn Werte aus der Datei *ein*. Zuerst wird ein Zeilenvorschub ausgeführt ($\backslash n$), dann wird eine Zahl gelesen (%g). Die gelesene Zahl wandert zur Variablen B und wird dort als Element B(1) abgespeichert. Der Vorgang „Zeilenvorschub-Zahl lesen“ wiederholt sich, bis zehn Zahlen gelesen sind. Die Datei darf hier in diesem Beispiel jedoch nur Zeilenvorschübe und Zahlen (diese eventuell mit führenden Blanks und Vorzeichen) enthalten.

Eingabe einer **Exceltabelle** erfolgt mit der Funktion `xlsread`. Die allgemeine Form ist:

```
Zielvariable = xlsread( Pfad , Bereich )
```

Beispiele: `MatrixS =xlsread('X:\C2\Hefe5.xlsx', 'A1:E55')`

```
VektorA =xlsread( Pfad, 'A1:A150')
```

Der Tabellenbereich A1:E55 im ersten Beispiel bzw. der Bereich A1:A150 im zweiten Beispiel sollte nur Zahlen und eventuell leere Zellen enthalten. Texte und andere Inhalte können zu einer Fehlermeldung führen. Lesen Sie auch die MATLAB-Hilfe zu `xlsread`.

Ist die **Länge** eines eingelesenen Vektors oder die **Zeilen- bzw. Spaltenzahl** einer eingelesenen Matrix unbekannt, besorgt man sich diese Information mit den Funktionen **length** und **size**.

Beispiel: Ein Vektor x und eine Matrix S liegen vor.

```
nx =length(x)      liefert die Anzahl der Elemente von Vektor x
```

```
nd=ndims(S)       liefert die Dimension von Matrix S, z.B. nd=2 bei einer Tabelle
```

```
[m, n]=size(S)    liefert z.B. Zeilenzahl m=5 und Spaltenzahl n=7
```

Beispiel 17:

<pre>%Ein/Ausgabe von Daten >>load f:\MATLABTestvektor.txt -ascii ← >>MATLABTestvektor ← MATLABTestvektor = 1 2 8 4 5 >>load f:\MATLABTestmatrix.txt -ascii ← >>MATLABTestmatrix ← MATLABTestmatrix = 44 77 33 -29 >>f=[7 3 9 5 6 2 3; 66 33 22 127 -44 0 3.14159265] ←</pre>	<p>Die Datei <i>MATLABTestvektor.txt</i> sei auf einem Stick im Laufwerk F:</p> <pre>1 2 8 4 5</pre> <p>Eine weitere Datei sei auf dem Stick. Ihr Name ist <i>MATLABTestmatrix.txt</i></p> <pre>44 77 33 -29</pre> <p>Wir wollen eine Matrix auf den Stick ausgeben. Dazu belegen wir Matrix f mit Zahlen. Matrix f hat 2 Zeilen.</p>
--	---

<pre> f = 7.0000 3.0000 9.0000 5.0000 6.0000 2.0000 3.0000 66.0000 33.0000 22.0000 127.0000 -44.0000 0 3.1416 >>save f:\MatrixFdaten.txt f -ascii ← >>load f:\MatrixFdaten.txt -ascii ← >>MatrixFdaten ← MatrixFdaten = 7.0000 3.0000 9.0000 5.0000 6.0000 2.0000 3.0000 66.0000 33.0000 22.0000 127.0000 -44.0000 0 3.1416 >>for i=1:10; A(i)=exp(sqrt(i)); end; ← A ← A = Columns 1 through 7 2.7183 4.1133 5.6522 7.3891 9.3565 11.5824 14.0940 Columns 8 through 10 16.9188 20.0855 23.6243 >>[Dateiname, Pfadname, Filternummer]=uiputfile('*.txt'); ← >>aus=fopen(strcat(Pfadname,Dateiname), 'wt'); ← >>for i=1:10; fprintf(aus, '\n%g',A(i));end; ← >>fclose(aus); ← % Wir lesen die Daten auf Matrix B ein % Filedialog >>[Dateiname, Pfadname, Filternummer] = uigetfile('*.txt'); ← >>Pfad=strcat(Pfadname, Dateiname); ← >>ein=fopen(Pfad, 'rt'); ← >>B=fscanf(ein, '\n%g',10); ← >>B ← B = 2.7183 4.1132 23.6243 >>B=B*2 ← </pre>	<p>Wir lassen uns die Matrix f anzeigen.</p> <p>Wir geben die Matrix mit Kommando save auf eine Datei <i>MatrixFdaten.txt</i> auf den Stick im Laufwerk F: aus. Die Datei besteht aus zwei Zeilen. Wir lesen sie wieder ein.</p> <pre> 7.0000000e+000 3.0000000e+000 9.0000000e+000 5.0000000e+000 6.0000000e+000 2.0000000e+000 3.0000000e+000 6.6000000e+001 3.3000000e+001 2.2000000e+001 1.2700000e+002 - 4.4000000e+001 0.0000000e+000 3.1415927e+000 </pre> <p>Wir belegen einen Vektor A mit Zahlen. Wir wollen die Werte auf A sehen.</p> <p>Wir öffnen Datei <i>MatrixA.txt</i> für Textausgabe im Ordner X:\C2.</p> <p>Wir schreiben in einer for-Schleife die zehn Elemente auf die Datei. Wir schließen die Datei. Wir schauen uns die Datei X:\C2\MatrixA.txt mit dem Editor an:</p> <pre> leere Zeile 2.71828 4.11325 23.6243 </pre> <p>Wir öffnen eine Eingabedatei mit ,rt'. Wir lesen 10 Zahlen nach Vektor B.</p> <p>Wir wollen uns Vektor B ansehen.</p> <p>Wir wollen sehen, ob man mit den Werten von B wirklich rechnen kann. Wir multiplizieren mit 2.</p>
--	---

<pre> >>B = 5.4366 8.2265 47.2486 >>fclose(ein); ← >>% Eingabe einer Exceltabelle ← >>TD=xlsread('X:\C2\Aufgabe5.xlsx', 'B2:E5'); ← >>TD ← TD = 2.0000 0.3500 NaN NaN 1.0000 1.4000 0.8500 0.3000 6.0000 0.4400 1.2500 0.6700 2.0000 0.6300 NaN NaN >>mn=size(TD); ← >>m=mn(1); ← >>m ← m = 4 </pre>	<p>Tatsächlich: Alle Werte sind verdoppelt.</p> <p>Wir schließen die Eingabedatei</p> <p>Wir erzeugen selbst eine Exceltabelle unter dem Namen Aufgabe5.xlsx mit vollen und einigen leeren Zellen. Wir lesen aus ihr die Zellen B2 : E5 .</p> <p>Wir schauen uns Matrix TD an. Vier leere Felder wurden gelesen (NaN = not a number).</p> <p>Wir besorgen uns Zeilenzahl und Spaltenzahl von TD als kleinen Vektor mn. Daraus extrahieren wir die Zeilenzahl m.</p>
---	---

1.13 Vergleichsoperatoren und logische Operatoren

Vergleichsoperatoren liefern einen logischen Wert 0 / 1 bzw. TRUE / FALSE.

<	kleiner	z.B.	$x < y$
>	größer		$x > y$
<=	kleiner oder gleich		$x \leq y$
>=	größer oder gleich		$x \geq y$
==	ist gleich		$x == y$
~=	ungleich		$x \neq y$

Werden Vektoren gleicher Länge verglichen, wird der verlangte Vergleich für jedes Elementepaar durchgeführt und die Vergleichsergebnisse als Nullen oder Einsen im Ergebnisvektor gespeichert.

isequal Diese Vergleichsfunktion für Felder liefert nur ein Ergebnis insgesamt. Eine 1, wenn die beiden Felder Element für Element gleich sind, sonst 0.

Logische Größen können mit **logischen Operatoren** weiter verknüpft werden:

&	logisches UND	$a \& b$ liefert nur eine 1, wenn $a=1$ und $b=1$ ist
	logisches ODER	$a b$ liefert nur eine 0, wenn $a=0$ und $b=0$ ist (Das Zeichen ist als dritte Belegung auf der < Taste. Tastenkombination „AltGr <“)
~	Negation	ist $a=1$, dann ist $\sim a=0$ ist $a=0$, dann ist $\sim a=1$

Beispiel 18:

<pre>>>V=[1:5] ←↵ V = 1 2 3 4 5 >>W=[2 2 7 4 5] ←↵ W = 2 2 7 4 5 >>b=(V==W) ←↵ b = 0 1 0 1 1 >>c=isequal(V,W) ←↵ c = 0 >>d=isequal(V,V) ←↵ d = 1 >>(2<3) & (4<1) ←↵ ans = 0 >>(2<3) & (1<4) ←↵ ans = 1 >>(2<3) (4<1) ←↵ ans = 1 >>~(2<3) ←↵ ans = 0</pre>	<p>Wir belegen einen Vektor V mit irgendwelchen Zahlen.</p> <p>Dann einen Vektor W.</p> <p>Wir testen auf Gleichheit der einzelnen Elemente. Die Antworten auf Vektor b fallen unterschiedlich aus. Z.B. ist V_1 ungleich W_1, aber V_2 gleich W_2 usw.</p> <p>Wir vergleichen die Vektoren V und W insgesamt. Die Antwort nach der Frage „ist gleich?“ fällt negativ aus. Die Vektoren V und W sind nicht gleich. Aber V ist mit sich selbst gleich.</p> <p>Wir Verknüpfen zwei Vergleichsergebnisse mit UND. Es ist FALSCH, dass $2<3$ und gleichzeitig $4<1$ ist. Aber es ist WAHR, dass $2<3$ und gleichzeitig $1<4$ ist.</p> <p>Zumindest ein Vergleich liefert ein WAHR. Damit ist das Ergebnis der ODER-Verknüpfung ebenfalls WAHR.</p> <p>$2<3$ ist WAHR. Die Negation (\sim) davon ist FALSCH.</p>
--	---

1.14 Zuweisungen

Zuweisungen weisen einer Variablen einen Wert zu. Es werden Zahlen oder Texte transportiert und auf einem Speicherplatz abgelegt. Der alte Inhalt der Variablen wird überschrieben und ist weg. Der Wert einer Variablen kann beliebig oft verwendet werden, z.B. in anderen Zuweisungen. Dabei ändert sich der Wert nicht.

Beispiel 19:

<pre>>>SQ=3177.234; mittel=19.321; n=5; ←↵ >>sigma=sqrt((SQ-n*mittel^2)/(n-1)) ←↵ sigma = 18.1020 >>x=3; y=2; ←↵ >>b=(x<5)&(y>0) ←↵ b = 1</pre>	<p>Erst eine Zuweisung ohne Semikolon löst nach der ENTER-Taste die Berechnung aus.</p> <p>Wir belegen Variable x und y und führen zwei Vergleiche durch, deren Ergebnisse wir mit UND verknüpfen. Das Ergebnis steht auf Variable b und ist WAHR.</p>
--	--

1.15 Verzweigungen

Das Programm kann je nach berechneten Zwischenergebnissen unterschiedliche Programmzweige durchlaufen. Es gibt verschiedene Formen der Verzweigung. Die einfachste Form besteht nur aus einer Bedingung und einer Anweisungsgruppe, die im Ja-Fall durchlaufen wird. Im Nein-Fall wird die Anweisungsgruppe übergangen (nicht ausgeführt).

if Bedingung; Anweisungsgruppe; **end**

Die Anweisungsgruppe wird nur durchlaufen, wenn die Bedingung den logischen Wert WAHR liefert (bzw. die Bedingung einen Wert ungleich 0 hat).

Eine Verzweigung mit Ja- und Nein-Fall hat die folgende allgemeine Form:

if Bedingung; Anweisungsgruppe 1; **else** Anweisungsgruppe 2; **end**

Die Anweisungsgruppe 1 wird nur durchlaufen, wenn die Bedingung den logischen Wert WAHR liefert (bzw. einen Wert ungleich 0 hat), ansonsten wird Anweisungsgruppe 2 durchlaufen. Es wird demnach entweder Anweisungsgruppe 1 (Ja-Fall) durchlaufen oder aber die Anweisungsgruppe 2 (Nein-Fall).

Tritt im Nein-Fall eine zusätzliche Bedingung auf, kann man in MATLAB die folgende if-Anweisung nehmen:

if Bedingung 1; Anweisungsgruppe 1; **elseif** Bedingung2; Anweisungsgruppe 2; **end**

Die Anweisungsgruppe 1 wird nur durchlaufen, wenn die Bedingung 1 den logischen Wert WAHR liefert (bzw. einen Wert ungleich 0 hat). Anweisungsgruppe 2 wird durchlaufen, wenn Bedingung 1 FALSCH liefert, aber Bedingung 2 WAHR.

Beispiel 20:

```
>>x=7; y=3; ←|
>>if x^2+y^2<=1; z=x^2+y^2; ←|
>>elseif x^2+y^2<=4; z=1; ←|
>>else z=sqrt(x^2+y^2)-1; end ←|
>>z ←|
z = 6.6158

>>if x^2+y^2<=1; z=x^2+y^2; ←|
>>else if x^2+y^2<=4; z=1; ←|
>>else z=sqrt(x^2+y^2)-1; end, end ←|
>>z ←|
z = 6.6158

>>if x>y; z=x^2; end ←|
>>z ←|
z = 49
```

Wir setzen x und y . Wenn $x^2+y^2 \leq 1$ ist, dann ergibt sich z als $z = x^2+y^2$. Wenn nicht und es gilt $x^2+y^2 \leq 4$, dann $z=1$. Wenn nicht $x^2+y^2 \leq 1$ und auch nicht $x^2+y^2 \leq 4$, dann wird $z = \sqrt{x^2 + y^2} - 1$.

Man kann denselben Sachverhalt auch ohne das „elseif“ programmieren. Der Nein-Fall nach dem **else** beginnt mit einer **if**-Anweisung, die ebenfalls einen Ja- und einen Nein-Zweig enthält. Wir benötigen jedoch zweimal das „end“ durch Komma getrennt, sozusagen die „Klammer zu“ nach einem **if** bzw. einem **else**.

Wenn $x > y$ ist, soll z den Wert $z = x^2$ erhalten. Andernfalls bleibt z unverändert, d.h. der alte

<pre>>>if x<y; z=x^2; else z=y^3; end ←↵ >>z ←↵ z = 27</pre>	<p>Wert von z bleibt erhalten.</p> <p>Wenn $x < y$ ist, wird $z = x^2$, andernfalls, d.h., wenn $x \geq y$ ist, wird $z = y^3$. Hier ändert z in jedem Falle seinen Wert.</p>
---	--

Die **switch-Anweisung** ist eine Mehrfachverzweigung. Man nimmt sie, wenn nicht nur ein Ja- und ein Neinfall vorliegt, sondern beliebig viele Fälle, die z.B. durchnummeriert sind. Man könnte die switch-Anweisung aber auch durch viele einfache if-Anweisungen realisieren. Lediglich der „sonstige Fall“ wird dann etwas aufwendiger zu programmieren.

Beispiel 21:

<pre>>> x=7, y=3, z=x^2+y^3 ←↵ x = 7 y = 3 z = 76 >> typnum=4; ←↵ >> switch typnum ←↵ >> case 1 ←↵ >> z=sqrt(x^2+y^4); ←↵ >> case 2 ←↵ >> z=pi*exp(-x/y); ←↵ >> otherwise ←↵ >> z=x+y+pi; ←↵ >> end ←↵ >> z ←↵ z = 13.1416</pre>	<p>Trennt man Anweisungen durch Kommas, dann zeigt MATLAB automatisch alle Werte an.</p> <p><i>typnum</i> sei eine Fallnummer für die folgende switch-Anweisung</p> <p>Fall 1: z ist die Wurzel aus $x^2 + y^4$.</p> <p>Fall 2: $z = \pi e^{-x/y}$.</p> <p>alle sonstigen Fälle: $z = x + y + \pi$.</p> <p>Ende der switch-Anweisung. Wir wollen den Wert von z wissen.</p>
---	--

1.16 Schleifen

Schleifen benötigt man für das mehrmalige Durchlaufen eines Programmstücks, z.B. bei Summenbildungen oder Iterationen, d.h. dem sich Annähern an eine richtige Lösung. Schleifen können mit **break** vorzeitig verlassen werden. Es gibt unterschiedliche Formen der Schleife:

Die **for-Schleife** nimmt man bevorzugt dann, wenn die Anzahl der Durchläufe bekannt ist, z.B. beim Aufaddieren von Vektorelementen, deren Anzahl feststeht.

Die **while-Schleife** nimmt man dann, wenn die Bedingung für einen erneuten Schleifendurchlauf sich erst beim Durchlaufen der Schleife ergibt.

Die allgemeine Form einer for-Schleife ist

for Laufgröße = Startwert : Endwert ; Anweisungsgruppe ; **end**

Ist die verlangte Schrittweite der Laufgröße ungleich 1, dann nimmt man die folgende Form:

for Laufgröße = Startwert : Schrittweite : Endwert ; Anweisungsgruppe ; **end**

Beispiel 22:

```

>>S=0; ←
>>for k=1 : 10; S=S+1/k; end; ←
>>S ←
    S = 2.9290

>>S=0; T=0; ←
for i=1 : 20; S=S+i^2; T=T+1/(1+i); end;
>>S ←
    S = 2870
>>T ←
    T = 2.6454

>>% Wurzel iterativ nach Newton ←
>>a=29; ←
>>epsilon=1e-9; ←
>>x=1; %Startwert ←
>>% ←
>>while abs(x^2-a)>epsilon; x=(x+a/x)*0.5;
                                end; ←
>>x ←
    x = 5.3852

>>x=sqrt(a) ←
    x = 5.3852

>>a=77665544.332211 ←
    a = 7.7666e+007

>>x=1; ←
>>while abs(x^2-a)>epsilon; k=k+1; ←
>>if k>20; break; end; ←
>>x=(x+a/x)*0.5; end; ←
>>x ←
    x = 8.8128e+003
>>k ←
    k = 18
>>sqrt(77665544.332211) ←
    ans = 8.8128e+003

>>% Variable vor der Schleife nennen ←
>>A=0; ←

>>%Schachtelung von Schleifen ←

```

Summen immer erst 0 setzen!

k läuft von 1 bis 10. Die jeweiligen Werte 1/k werden zur Summe S addiert.

$$\text{Summe } S = \sum_{k=1}^{10} (1/k).$$

Gleichzeitig 2 Summen bilden.

$$S = \sum_{i=1}^{20} (i^2) \quad \text{und}$$

$$T = \sum_{i=1}^{20} (1/(1+i))$$

Iterative, d.h. sich annähernde, Berechnung einer Quadratwurzel nach dem Newton-Verfahren. Wir wollen die Wurzel aus a=29 ziehen. Die Wurzel soll x heißen. Man startet mit einem beliebigen positiven Wert, z.B. x=1. In einer while-Schleife werden immer bessere Lösungen gefunden. Die Schleife wird nochmals durchlaufen, wenn $|x^2-a| > 10^{-9}$ ist, d.h. die Wurzel quadriert ergibt noch nicht exakt genug den Wert a.

Zur Kontrolle berechnen wir die Wurzel noch einmal mit dem professionellen Wurzelprogramm von MATLAB.

Eine Schleife mit Schleifendurchlaufzähler k und Abbruch der Schleife nach k>20 Durchläufen. Wir wollen die Wurzel aus einer recht großen Zahl a ziehen.

Wir starten wieder mit x=1.

Variable k zählt die Schleifendurchläufe. Wenn k>20 ist, beenden wir die Schleife.

Ansonsten iterieren wir wie oben nach Newton. Wir wollen die berechnete Wurzel sehen. Die Wurzel ist $x=8,8128 \cdot 10^3$.

Wir wollen den Schleifenzähler sehen. 18 Iterationen wurden benötigt.

Zur Kontrolle die Berechnung mit dem professionellen Wurzelprogramm von MATLAB.

Besonders for-Schleifen treten oft **ineinandergeschachtelt** auf, nämlich dann, wenn mehrdimensionale Matrizen zu bearbeiten sind.

<pre>>>for i=1:3; ←␣ >>for k=1:4; ←␣ A(i,k)=exp(-0.1*(i^2+k^2)); ←␣ >>end; ←␣ >>end; ←␣ >>A ←␣ A = 0.8187 0.6065 0.3679 0.1827 0.6065 0.4493 0.2725 0.1353 0.3679 0.2725 0.1653 0.0821</pre>	<p>Die äußere, langsame Schleife lässt i laufen. Die innere, schnelle Schleife lässt k laufen. Wir belegen Matrixelement $A_{ik} = e^{-0,1(i^2+k^2)}$ Das Ende des k-Schleifenkörpers. Das Ende des i-Schleifenkörpers. Wir wollen die Matrix A sehen: Matrix A hat 3 Zeilen gemäß $i = 1 : 3$ und 4 Spalten gemäß $k = 1 : 4$</p>
---	---

In den folgenden Programmbeispielen werden der Prompt >> und das Anzeigen der Entertaste ←␣ weggelassen.

Aufgabe 2: Reihenentwicklung und einfache Integration

Aufgabenstellung: Programmieren Sie das Gaußsche Fehlerintegral als Reihe und integrieren Sie es numerisch. Bilden Sie für jeden berechneten Wert den relativen Fehler zwischen beiden Berechnungsarten.

Gruppe 1-9	Gruppe 10-18	Gruppe 19-27	Gruppe 28-36
Bestimmen Sie ungefähr den Wert x, bei dem der relative Fehler größer 0,5% wird.	Bestimmen Sie ungefähr den Wert x, bei dem der relative Fehler größer 0,9% wird.	Bestimmen Sie ungefähr den Wert x, bei dem der relative Fehler größer 1,2% wird.	Bestimmen Sie ungefähr den Wert x, bei dem der relative Fehler größer 1,5% wird.
Geben Sie Fr, Fi und sigma mit 12 Druckstellen, davon 7 Dezimalen, aus.	Geben Sie Fr, Fi und sigma mit 11 Druckstellen, davon 6 Dezimalen, aus.	Geben Sie Fr, Fi und sigma mit 10 Druckstellen, davon 5 Dezimalen, aus.	Geben Sie Fr, Fi und sigma mit 12 Druckstellen, davon 5 Dezimalen, aus.

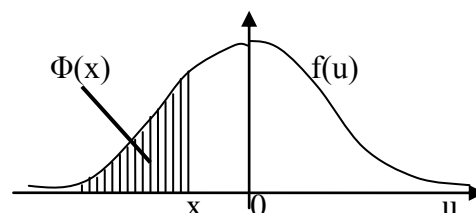
Ergebnisse: Das Textfile *Aufgabe2.txt* zur Aufgabe und die Ausgabedatei *Aufgabe2Integraldaten.txt* .

2.1 Einführung in die Reihenentwicklung des Fehlerintegrals

Das Gaußsche Fehlerintegral lautet

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-u^2/2} du .$$

$\Phi(x)$ ist die Fläche von $-\infty$ bis x unter der Gaußschen Glockenkurve.



Die Werte von $\Phi(x)$ werden beim u-Test in der mathematischen Statistik (Datenauswertung) benötigt. Auch der Binomialtest führt bei großen Zahlen auf dieses Integral.

Bei der Reihenentwicklung gehen wir von der Reihe der e-Funktion $f(x) = e^x$ aus.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots \quad \text{für } n \rightarrow \infty$$

Wir ersetzen x durch $-u^2/2$ und erhalten die Reihenentwicklung von $e^{-u^2/2}$.

$$e^{-u^2/2} = 1 - \frac{u^2}{2 \cdot 1!} + \frac{u^4}{2^2 \cdot 2!} - \frac{u^6}{2^3 \cdot 3!} + \dots (-1)^n \frac{u^{2n}}{2^n \cdot n!} + \dots \quad \text{für } n \rightarrow \infty$$

Wir integrieren gliedweise nach der Potenzformel, ersetzen das u durch ein x und erhalten die Reihenentwicklung:

$$F(x) = x - \frac{x^3}{3 \cdot 2 \cdot 1!} + \frac{x^5}{5 \cdot 2^2 \cdot 2!} - \frac{x^7}{7 \cdot 2^3 \cdot 3!} + \dots (-1)^n \frac{x^{2n+1}}{(2n+1) \cdot 2^n \cdot n!} + \dots \quad \text{für } n \rightarrow \infty$$

Da diese Reihenentwicklung von der Null aus startet, erhalten wir für $x \rightarrow \infty$ nur den halben Wert. Der Zusammenhang von $F(x)$ mit $\Phi(x)$ ist

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} (F(x) + 0,5).$$

Bei der Berechnung einer Reihe im Computer nutzt man die Tatsache aus, dass sich das n -te Glied der Reihe aus dem $n-1$ -ten Glied meist auf einfache Weise berechnen lässt. Bei der $F(x)$ -Reihe sieht man, dass wir immer das nächste Glied erhalten, wenn wir das vorangehende

Glied mit dem Faktor

$$-\frac{x^2(2n-1)}{(2n+1) \cdot 2 \cdot n}$$

multiplizieren, wobei n die Nummer des gerade zu berechnenden Gliedes ist. Die Nummerierung läuft von $n = 0, 1, 2, \dots$. D.h., wenn wir die Reihe $F(x)$ berechnen, starten wir mit Glied $g_0 = x$. Wir setzen $n=1$ und multiplizieren mit obigem Faktor und erhalten das nächste Glied der Reihe $g_1 = -x^3/(3 \cdot 2 \cdot 1)$. Wir erhöhen n auf $n=2$, multiplizieren g_1 mit obigem Faktor und erhalten Glied $g_2 = x^5/(5 \cdot 2 \cdot 2 \cdot 1 \cdot 2)$. Auf diese Weise entsteht automatisch die Fakultät $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$ und die Zweierpotenz $2^n = 2 \cdot 2 \cdot \dots \cdot 2$ im Nenner der Glieder. Man beendet bei dieser Reihe die Summation der Glieder, wenn das Glied kleiner 10^{-16} ist. Dann leisten auch die restlichen Glieder bis Unendlich keinen nennenswerten Beitrag mehr zur Summe. Das Ganze lässt sich sehr schön und kurz in eine Programmschleife verpacken. Problematisch ist der Wert von x . Mathematisch konvergiert die Reihe für jedes x gegen einen festen Wert. Leider ist das im Computer nicht so. Für große x wachsen die Absolutbeträge der einzelnen Glieder für wachsendes n erst sehr stark an, um dann wieder abzunehmen. Wir bekommen massive Probleme mit der Genauigkeit aufgrund der endlichen Zahlenlänge im PC.

Deshalb ist ein Teil dieser Aufgabe, dass Sie ungefähr den Wert x bestimmen, bei dem der relative Fehler größer als 1% wird (>0.01). Der relative Fehler σ (sigma) berechnet sich hier nach folgender Formel:

$$\sigma(x) = \frac{2 \cdot |\Phi_{\text{Reihe}}(x) - \Phi_{\text{Integral}}(x)|}{\Phi_{\text{Reihe}}(x) + \Phi_{\text{Integral}}(x)}.$$

2.2 Beispielprogramm für Reihe und Integral

Die Sinusreihe lautet:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \dots \quad \text{für } n \rightarrow \infty$$

Das Glied g_n ergibt sich aus Glied g_{n-1} durch Multiplikation mit dem Faktor $-\frac{x^2}{(2n)(2n+1)}$.

Glied g_0 ist $g_0=x$. Laufindex n läuft im Programm von $n=1, 2, 3, \dots$

Die Sinusreihe ist gleichzeitig die Reihe für $\int \cos(x)dx$, denn es gilt $\sin(x) = \int \cos(x)dx$.
Da $\sin(x)$ auch Null werden kann, funktioniert unser Abbruch nach dem relativen Fehler σ (sigma) nicht. Wir benutzen hier deshalb den absoluten Fehler, d.h. $\sigma(x) = |Fr(x) - Fi(x)|$,
d.h. den Betrag der Differenz von Reihe $Fr(x)$ und Integral $Fi(x)$.

Beachten Sie, dass MATLAB bei der Übermittlung von Standardfunktionsnamen als Parameter das Zeichen @ benutzt, z.B. $Fi = \text{quad}(@\cos, 0, x)$.
Bei der Übermittlung des Namens selbst geschriebener Funktionen wird der Funktionsname jedoch in Apostrophe gesetzt, z.B. $Fi = \text{quad}(\text{'meineFunktion'}, 0, x)$

<pre> aus=fopen('X:\C2\Aufgabe2Integraldaten.txt', 'wt'); fprintf(aus, ' x Reihe Fr(x) Integral Fi(x) Fehler'); sigma=0; x=0.1; while sigma<0.01; Fr=x; g=x; n=0; while abs(g)>1e-15; n=n+1; g=g*(-x^2/((2*n)*(2*n+1))); Fr=Fr+g; end; % Ende while g>1e-15; Fi=quad(@cos, 0, x); sigma=abs(Fr-Fi); fprintf(aus, '\n% 10.1f % 10.5f % 10.5f % 10.5f ', x, Fr, Fi, sigma); x=x+0.1; end; % Ende while sigma<0.01; fclose(aus) </pre>	<p>Ausgabedatei öffnen für Textausgabe (das Handle heißt <i>aus</i>) Kopfzeile Ausgabedatei schreiben.</p> <p>Fehler ist erst mal 0 setzen. Wir starten hier mit $x=0.1$</p> <p>While-Schleife, die durchlaufen wird, solange $\text{sigma}<0.01$ ist.</p> <p>Startwert der Sinusreihe $Fr(x)$. Wert von Glied g_n für $n=0$. While-Schleife, die durchlaufen wird, solange das Glied $>10^{-15}$ ist. Wir erhöhen n um 1. Das neue Glied berechnen. Die Summe der Reihe bilden.</p> <p>Schleifenende der inneren Schleife.</p> <p>Numerische Integration des Kosinus von 0 bis x ergibt $\sin(x)$. Absoluter Fehler sigma.</p> <p>Ausgabe einer Zeile.</p> <p>x um 0.1 erhöhen.</p> <p>Schleifenende der äußeren Schleife.</p> <p>Ausgabedatei schließen</p>
--	---

Ein Blick in die Datei `X:\C2\Aufgabe2Integraldaten` zeigt uns folgendes Bild:

x	Reihe Fr(x)	Integral Fi(x)	Fehler
0.1	0.09983	0.09983	0.00000
0.2	0.19867	0.19867	0.00000

```

.....
28.3  -0.02567  -0.02566  0.00001
28.4  -0.12536  -0.12534  0.00003
.....
35.1  -0.52244  -0.51626  0.00618
35.2  -0.58811  -0.59918  0.01108

```

Hier bricht das Programm ab

2.3 Hinweise für Ihr Programm

Stellen Sie erst zuerst das Funktionsprogramm der Funktion $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ her.

Geben Sie der Funktion einen Phantasienamen, z.B. *Glockenkurve* oder *FunktionAufgabe2*. Sie benötigen diese Funktionsdefinition in der Anweisung `quad(...)` zum Integrieren aus dem Beispielprogramm. Sie ersetzt dort den Kosinus (*cos*) aus dem Beispielprogramm.

Das folgende Beispiel zeigt die Vorgehensweise am Beispiel der Funktion mit zwei Veränderlichen

$$KK(\theta, \alpha) = 3\theta^2 - 7\alpha^2$$

Die folgende Definitionsweise liefert ein dauerhaftes M-File in Ihrem C2-Ordner:

Minimieren Sie MATLAB → Start → Alle Programme → Zubehör → Editor

Jetzt im Editor die 3 Zeilen

```

function [w ] = KK(teta, alfa)
% Beispielfunktion programmiert von Mausli Musterfrau
[w]=3*teta.^2-7*alfa.^2; % beachte elementweises Potenzieren .^

```

eintippen und als Datei *KK.m* in den C2-Ordner Ihrer X-Platte (samba) speichern. Editor schließen. Nun müssen wir MATLAB noch am Beginn jeder Sitzung mitteilen, dass im Ordner C2 wichtige Programme vorhanden sind:

MATLAB starten bzw. maximieren → File → Set Path → Add Folder → Arbeitsplatz → X: (samba) → C2 → close → die Frage „Do you wish to save ...“ mit Nein beantworten. MATLAB weiß jetzt (zumindest in dieser Sitzung), dass Programme, z.B. Funktionen, auch im Ordner C2 zu suchen sind.

Des Weiteren ändert sich die Reihenentwicklung. Sie müssen das neue Glied der Reihe mit dem Faktor berechnen, der in Abschnitt 2.1 genannt wird. Außerdem müssen Sie gleich nach dem Ende der inneren Schleife den Wert *Fr* noch mit dem Faktor $1/\sqrt{2\pi}$ multiplizieren, etwa in der Form $Fr = Fr * (1./\text{sqrt}(2 * \text{pi}))$; (beachte elementweises Dividieren ./)

Ihre letzte Änderung betrifft die Berechnung des Fehlers *sigma*. Ihr Programm soll den relativen Fehler aus Abschnitt 2.1 benutzen. Dieser lässt sich jedoch nicht für $x=0$ berechnen, da dort $Fr=0$ und $Fi=0$ sind. Sie starten deshalb nicht mit $x=0$, sondern mit $x=0.1$.

Aufgabe 3: Komplexes Rechnen

Aufgabenstellung: a) Plotten Sie die **Ortskurve** $F_o(p)$ der Kreisschaltung eines Regelkreises. Suchen Sie durch probieren solche PID-Parameter, dass die Ortskurve ziemlich genau durch den Punkt -0.5 auf der reellen Achse geht.

b) Plotten Sie die **Analytische Landschaft** der Funktion $1+F_o(p)$

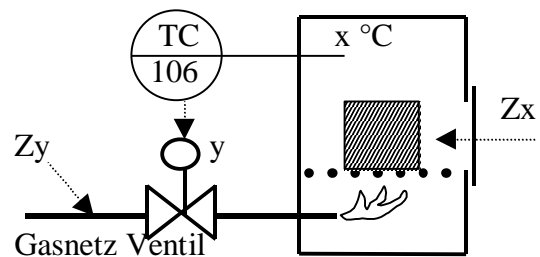
Gruppe 2-10 Sie rechnen mit	Gruppe 11-19 Sie rechnen mit	Gruppe 20-28 Sie rechnen mit	Gruppe 1, 29-36 Sie rechnen mit
$K_{PR}= 10,0$	$K_{PR}= 12,0$	$K_{PR}= 8,0$	$K_{PR}= 20,0$
$T_N = 17,5$	$T_N = 21,5$	$T_N = 15,5$	$T_N = 27,5$
$T_V = 3,5$	$T_V = 4,5$	$T_V = 3,0$	$T_V = 6,5$
$K_{PS}=16,2$	$K_{PS}=12,2$	$K_{PS}=14,0$	$K_{PS}=19,2$
$T = 8,5$	$T = 18,5$	$T = 12,5$	$T = 28,5$
$K_p= 1,0$	$K_p= 1,0$	$K_p= 1,0$	$K_p= 1,0$
$T = 2,7$	$T = 5,7$	$T = 3,7$	$T = 6,7$

Ergebnisse: Die beiden Graphiken *Aufgabe3Ortskurve.tif*,
Aufgabe3AnalytischeLandschaft.tif.

3.1 Kurze Einführung in die Regelung

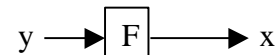
Beispiel Temperaturregelung eines Industrieofens

Die Graphik zeigt das RI-Fließbild des Ofens: Im Inneren herrscht die Temperatur x °C. Der Regler stellt ein Gasventil, um die Ist-Temperatur auf dem Sollwert zu halten. Stellgröße ist der Hub y am Gasventil. Störungen Z_x können durch die geöffnete Ofentür und kaltes Material entstehen, Störungen Z_y durch Druckschwankungen im Gasnetz.

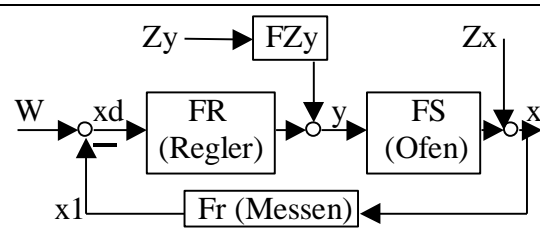


Die Übertragungsfunktion $F(p)$ eines Gliedes ist der Quotient von Ausgangssignal x und Eingangssignal y . $\mathbf{x(p)}$ heißt Laplace-Transformierte der Zeitfunktion $x(t)$, analog $y(p)$.

$$F(p) = \frac{x(p)}{y(p)}$$



Der Regelkreis wird vom Regler (Übertragungsfunktion F_R), der Regelstrecke (Übertragungsfunktion F_S) und der Rückführung der Istgröße (Übertragungsfunktion F_r) gebildet. Die Z_x -Störungen addieren sich direkt zur Istgröße, die Z_y -Störungen mittels Übertragungsfunktion F_{Zy} .



Blockschaltbild

W Sollwert, z.B. 1000 °C Schmiedetemperatur für Stahlrohlinge
 x tatsächliche Temperatur im Ofen

x_1	Istwert nach dem Messvorgang, z.B. mit einem Thermoelement NiCrNi gemessen
$x_d = W - x_1$	Regeldifferenz Sollwert minus Istwert
y	Stellgröße, z.B. der Hub des Gasventils in % von 100
Z_x	Störgröße, z.B. die Temperaturabsenkung in °C durch eine offene Ofentür oder einen kalten Stahlblock
Z_y	Störgröße, z.B. eine Druckminderung im Gasnetz. Z.B. könnten sich 10% Minderdruck wie 15% weniger Hub am Gasventil auswirken. Deshalb rechnet die Übertragungsfunktion F_{Zy} Druckschwankungen [mbar] in y -Werte [% Hub am Ventil] um.
FR	Übertragungsfunktion des Reglers, z.B. eines PID-Reglers
FS	Übertragungsfunktion der Regelstrecke einschließlich Stellantrieb (hier der Ofen und das Regelventil), modelliert als PT_2 -Strecke)
Fr	Übertragungsfunktion der Rückführung, z.B. die Verzögerung der Temperaturmessung durch eine Keramikummantelung des Thermoelements modelliert als PT_1 -Strecke

Regelstrecken sind Prozesse, bei denen Prozessparameter (z.B. Temperatur, Druck, pH, Trübung, Konzentration, Durchfluss usw.) geregelt werden. Wichtig ist die Reaktion des zu regelnden Prozessparameters auf seine Stellgröße, z.B. wie stark und wie schnell reagiert die Temperatur auf eine definierte Hubänderung am Heizventil. Regelstrecken werden nach der **Sprungantwort** klassifiziert. Eine Sprungantwort ist der Verlauf der Regelgröße x nach einer sprungartigen Erhöhung der Stellgröße y um einen definierten Betrag $\Delta y = U_0$.

	<p>Die Graphik zeigt einen Stellgrößensprung, z.B. das Einschalten einer Zusatzheizung. Die Größe U_0 muss keine Spannung sein, sondern ist die Änderung einer beliebigen Stellgröße (Heizleistung, Rührerdrehzahl, Stromstärke, ..)</p>
	<p>PT1-Sprungantwort (Ein-Speicher-Modell): Die Regelgröße steigt sofort nach dem Stellgrößensprung vom alten Beharrungswert x_0 und nähert sich asymptotisch als e-Kurve dem neuen Beharrungswert x_B. Beispiel: Temperatur im Rührkessel nach Heizsprung</p>
	<p>PTn-Sprungantwort (n-Speicher-Modell): Die Regelgröße steigt erst nach einer Verzugszeit vom alten Beharrungswert x_0 zum Wendepunkt und von da asymptotisch zum neuen Beharrungswert x_B. Beispiel: Temperatur am Duschkopf einer Dusche.</p>

Wir rechnen jedoch nicht mit den Zeitfunktionen $x(t)$, $y(t)$, sondern mit den Laplace-Transformierten $x(p)$ und $y(p)$. Die Laplace-Transformation fußt auf der Fourieranalyse, d.h. der Zerlegung einer beliebigen Funktion in unendlich viele Frequenzanteile, d.h. Sinus- und Kosinusschwingungen. Um z.B. den Anteil einer bestimmten Sinusschwingung mit Frequenz ω in einer Funktion $f(t)$ zu bestimmen, muss das Integral $\int_0^{\infty} f(t) \sin(\omega t) dt$ gelöst werden.

Nun gibt es leider viele Funktionen $f(t)$, die sich so nicht integrieren lassen. Dazu gehört schon die simple Funktion $f(t)=1$. Da der Wert ∞ der oberen Integralgrenze nicht definiert ist, gibt es auch keinen definierten Integralwert. Das hat Laplace erkannt und den Trick mit der Dämpfung der Funktion eingeführt. Lässt man $f(t)$ für $t \rightarrow \infty$ gegen Null gehen, dann ist das

Integral plötzlich lösbar, d.h. wir berechnen das Integral $\int_0^{\infty} f(t) \sin(\omega t) e^{-\delta t} dt$ mit einer Dämpfungskonstanten δ .

Funktion im Zeitbereich $x(t), y(t)$ **Laplace-Transformation** Funktion im Bildbereich $X(p), Y(p)$
 (meist mit Korrespondenztafeln) (auch Frequenzbereich genannt)

mit $p = \delta + j\omega$, wobei $\delta =$ beliebige dimensionslose Dämpfung der Funktion für $t \rightarrow \infty$ ist und $\omega =$ eine Kreisfrequenz zwischen 0 und ∞ ist. Der Bestandteil $e^{j\omega t}$ im Term e^{-pt} erzeugt uns die Sinus- und Kosinuswellen der Fourieranalyse. j ist die imaginäre Einheit. Damit ist p eine komplexe Zahl.

Die Rücktransformation vom Frequenzbereich in den Zeitbereich ist mathematisch sehr anspruchsvoll (Funktionentheorie) und wird fast immer mit den Korrespondenztafeln gemacht. Die folgende Tafel zeigt die **Korrespondenzen** für einige wenige ausgewählte Funktionen.

Modellbezeichnung	Modellgleichung im Zeitbereich	Übertragungsfunktion im Frequenzbereich
P	$x = K_p y$	$F(p) = K_p$
PT ₁	$T\dot{x} + x = K_p y$	$F(p) = \frac{K_p}{1 + Tp}$
PT _n	$a_n x^{(n)} + \dots + a_1 \dot{x} + a_0 x = b_0 y$	$F(p) = K_p \frac{1}{(1 + Tp)^n}$
PID	$x = K_p \left[y + \frac{1}{T_n} \int y(t) dt + T_v \dot{y} \right]$	$F(p) = K_p \left[1 + \frac{1}{T_n p} + T_v p \right]$

Links steht die Gleichung bzw. DGL des Übertragungsverhaltens von y nach x im Zeitbereich, rechts daneben die Laplacetransformierte $F(p)$ der Übertragungsfunktion $f(t) = x(t)/y(t)$. T sind Zeitkonstanten, K_p konstante Verstärkungen, a_i konstante Koeffizienten, x_0 ein Anfangswert, $x^{(n)}$ die n -te Ableitung von $x(t)$. T_n und T_v sind PID-Parameter. Wir sehen, die Formeln sind für jedes Modell (PT₁, PT_n, PID) bekannt. Man muss nur die richtigen Zahlenwerte für die Koeffizienten einsetzen.

Liegt eine Kreisschaltung vor wie beim Regelkreis, dann gilt die Übertragungsfunktion einer Kreisschaltung. Diese ist

$$x(p) = \frac{FR(p)FS(p)}{1 + FR(p)FS(p)Fr(p)} W(p)$$

Sie beschreibt den Einfluss des Sollwertverlaufs W auf die Temperatur x im Ofen.

3.2 Zahlenbeispiel für den Einfluss von Sollwertänderungen am Regelkreis

PID-Regler mit $K_{PR} = 15,0$ [% / K] Reglerverstärkung in % Hub pro Grad Abweichung
 $T_N = 7,5$ [s] Nachstellzeit des I-Anteils des PID-Reglers
 $T_V = 1,9$ [s] Vorhaltezeit des D-Anteils des PID-Reglers

und der Übertragungsfunktion

$$FR(p) = K_{PR} \left[1 + \frac{1}{T_N p} + T_V p \right] = 15,0 \left[1 + \frac{1}{7,5 p} + 1,9 p \right]$$

Ofen mit PT₂-Verhalten $K_{PS}=16,2$ [K / %] Streckenverstärkung in Grad pro % Ventilhub
 $T = 8,2$ [s] Zeitkonstante eines Verzögerungsgliedes
 und der Übertragungsfunktion

$$FS(p) = K_{PS} \left(\frac{1}{1 + pT} \right)^n = 16,2 \left(\frac{1}{1 + 8,2 \cdot p} \right)^2$$

Messung mit PT₁-Verhalten $K_p= 1,0$ [K/K] Verstärkungsfaktor in Grad pro Grad.
 $T = 2,7$ [s] Zeitkonstante der Messverzögerung
 und der Übertragungsfunktion

$$Fr(p) = \frac{K_p}{1 + Tp} = \frac{1}{1 + 2,7 p}$$

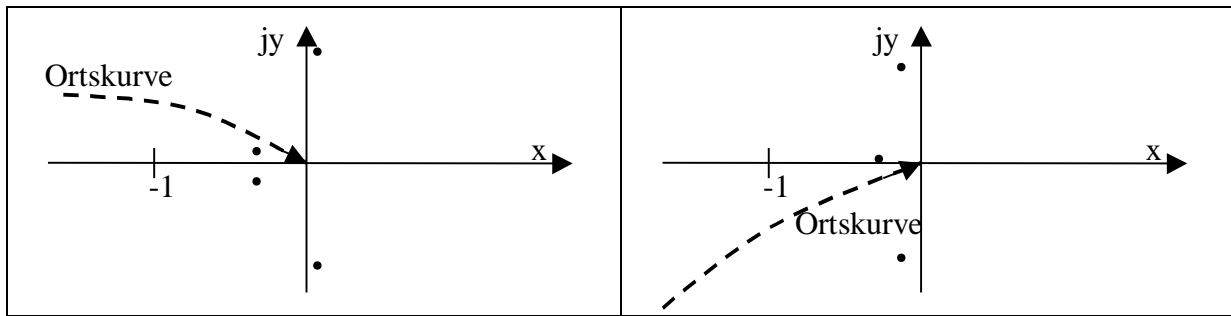
3.3 Zwei gleichwertige Stabilitätskriterien

Stabilität in der Regelung heißt, dass keine Dauerschwingungen auftreten, und auch kein Aufschaukeln von Schwingungen möglich ist. Für die Beurteilung der Stabilität eines Regelkreises ist vor allen Dingen der Nenner der Übertragungsfunktion $1+FR(p) \cdot FS(p) \cdot Fr(p)$ wichtig. Das Produkt $F_0(p)=FR(p) \cdot FS(p) \cdot Fr(p)$ heißt *Übertragungsfunktion des aufgeschnittenen Regelkreises* und spielt eine wichtige Rolle. $F_0(p)$ beschreibt die Reihenschaltung von Regler, Regelstrecke und Rückführung (siehe Blockschaltbild oben).

Das **Wurzelort-Kriterium** der Stabilität (Root Locus Criterion): Sind die Realteile aller Wurzeln (Nullstellen) der Gleichung $1+F_0(p)$ negativ, dann treten nur gedämpfte Schwingungen auf, und die Regelung ist stabil.

Das **Ortskurven-Kriterium** (auch Nyquist- oder Routh-Hurwitz Criterion) : Lässt die Ortskurve $F_0(j \omega)$ für $\omega \rightarrow \infty$ den Punkt "-1" auf der reellen Achse "links liegen", dann ist keine Selbstverstärkung möglich und die Regelung ist stabil.

<p>Eine Lösung mit den PID-Parametern $K_{PR}=30$, $T_N=8$ und $T_V=1,5$ zeigt zwei konjugiert komplexe Wurzeln im positiven Bereich bei $0,03 \pm 1,93 j$ und zwei konjugiert komplexe Wurzeln im negativen Bereich bei $-0,33 \pm 0,15 j$. Die Ortskurve lässt den Punkt "-1" rechts liegen. Beide Kriterien sagen instabil.</p>	<p>Eine Lösung mit den PID-Parametern $K_{PR}=3$, $T_N=30$ und $T_V=7,5$ zeigt alle Wurzeln im negativen Bereich. Die Ortskurve lässt den Punkt "-1" links liegen. Beide Kriterien sagen stabil. Leider ist diese Lösung nicht optimal. Der Regler reagiert zu langsam und zu schwach.</p>
--	---



Wie berechnet man die Ortskurve einer Übertragungsfunktion? Die Ortskurve einer komplexen Funktion $F(p)$ entsteht, wenn man nacheinander p -Werte in die Funktion einsetzt, die auf der imaginären Achse der komplexen Zahlenebene liegen, d.h. $p=0+\omega j$. Lässt man z.B. ω Werte von 1 bis 10,0 in Schritten von 0,01 durchlaufen, entstehen 1000 komplexe $F(p)$ -Werte $z = x+yj$. Diese Werte lassen sich in MATLAB als 2D-Plot plotten.

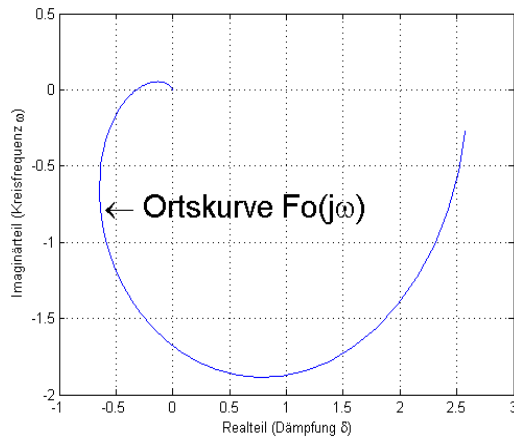
Wie berechnet man die Wurzeln (Nullstellen) einer komplexen Funktion $F(p)$? Systeme wie MATLAB bieten da verschiedene Lösungen an. Eine Methode ist die folgende: Man nimmt den Bildschirm als Gaußsche komplexe Zahlenebene und steckt den Zahlenbereich für das komplexe Argument $p=\delta+\omega j$ ab, z.B. den Realteil $-3 \leq \delta \leq 3$ und den Imaginärteil $-3 \leq \omega \leq 3$. Auf diese Weise kann man jedem Bildpunkt in diesem abgesteckten Bereich einen komplexen Wert p zuordnen. Die p -Werte setzt man in die komplexe Funktion $F(p)$ ein, berechnet deren Werte $z = x+yj$. Dann bildet man den Betrag $\text{abs}(z)$, d.h.

$$|F(p)| = |z| = \sqrt{x^2 + y^2}.$$

Die reelle Funktion $|F(p)|$ nennt man *analytische Landschaft* der komplexen Funktion $F(p)$. Nullstellen könnte man z.B. in einer 3D-Darstellung finden. Laut Theorie gibt es in einer analytischen Landschaft keine Hochtäler, d.h., jedes Tal hat im Zentrum eine Nullstelle. Eine andere Methode wäre, die Nullstellen von $|F(p)|$ direkt mit numerischen Verfahren zu suchen, z.B. mit einem Gradientenverfahren. Man startet von einem zufällig ausgewählten Punkt p auf der Gaußschen Zahlenebene. Das Gradientenverfahren sucht die Richtung des steilsten Abstiegs und verändert p so, dass es in diese Richtung läuft. Irgendwann ist eine Nullstelle erreicht. Das Problem ist, dass man nicht weiß, ob man alle Nullstellen gefunden hat.

3.4 Beispiel der Berechnung einer Ortskurve

<pre>% Ortskurve mit Beschriftung p=[0.001:0.001:10]*1i; % Beachte die punktweisen Operationen .* ./ .^ f=12*(1+2*p).*(0.6./(1+35*p)).^3.*(1./(1+0.9*p)).^2; plot(f); grid; xlabel(' Realteil (Dämpfung \delta)'); ylabel(' Imaginärteil (Kreisfrequenz \omega)'); text(-0.6, -0.75, '\leftarrow Ortskurve Fo(j\omega)', 'FontSize', 22);</pre>	<p>Kommentar. Herstellung der 10.000 komplexen p-Werte (alle rein imaginär). Berechnung aller 10.000 komplexen Funktionswerte $f(p)$. Herstellung des Plots. Gitter zeichnen lassen. x-Achsenbeschriftung x-Achsenbeschriftung Schriftzug <i>Ortskurve</i> und linker Pfeil (<code>leftarrow</code>).</p>
---	---



Die Ortskurve zeigt graphisch, wie sich ein sinusförmiges Signal beim Durchgang durch ein Übertragungsglied mit der Übertragungsfunktion $F_0(p)$ verändert. Jeder Punkt der Ortskurve ist einer Kreisfrequenz ω zugeordnet. Zieht man vom Koordinatenursprung $0+0i$ eine Linie zu einem beliebigen Punkt der Ortskurve, dann gibt die Linienlänge die Amplitude des Ausgangssignals. (Die Amplitude des Eingangssignals wird mit 1 angenommen.) Der Winkel, den die gezogene Linie mit der positiven reellen Achse bildet, ist der Phasenwinkel, d.h. das zeitliche Nachhinken des Ausgangssignals hinter dem Eingangssignal. Ein Winkel von 90° (entsprechend $\pi/2$) entspricht einer $1/4$ Wellenperiode, entsprechend ein Winkel von 180° (oder π) einem Nachhinken um eine halbe Periodendauer.

Im Allgemeinen nimmt die Amplitude für höhere Frequenzen ab und der Phasenwinkel zu, da die Schaltung dem Eingangssignal nicht genügend schnell folgen kann.

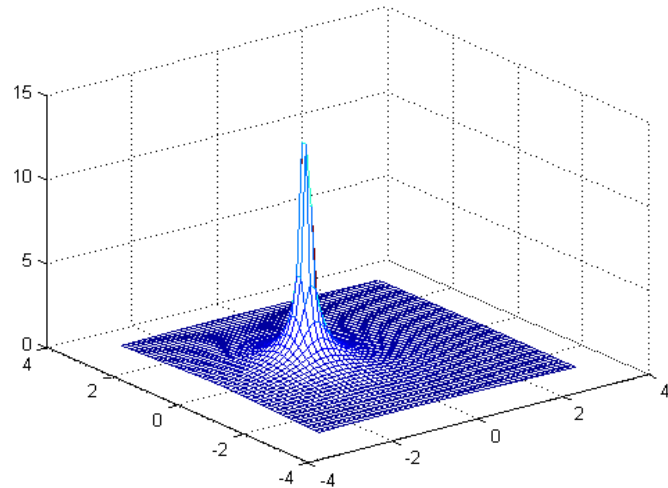
3.5 Beispiel der Berechnung einer analytischen Landschaft

Die Beispielfunktion lautet hier:
$$F(z) = \frac{1}{1+z}$$

<pre>% 3-D Plot analytische Landschaft [X,Y]=meshgrid(linspace(-3, +3, 50), linspace(-3, +3, 50)); kZ=X+Y.*1i; % Beachte elementweise Operation .* % Beachte elementweise Operation ./ Z=abs(1./(1+kZ)); mesh(X,Y,Z)</pre>	<p>Kommentar. Herstellung eines Kreuzgitters: Zuerst die 50 Punkte der reellen Achse x von -3 bis 3, dann die der imaginären Achse (y-Achse). Herstellung einer komplexen Zahl kZ an allen $50 \cdot 50 = 2500$ Gitterpunkten. „1i“ ist die imaginäre Einheit. kZ ist die Koordinate des Gitterpunktes auf der Gaußschen Zahlenebene. Berechnung der reellen Beträge $Z = 1/(1+kZ)$ der komplexen Funktionswerte $1/(1+kZ)$ an allen 2500 Gitterpunkten. Zeichnen des 3-D Plots.</p>
--	---

Sie müssen natürlich für Ihre analytische Landschaft **Ihre Funktion** $F(p) = 1 + F_0(p)$ verwenden.

Wir sehen, MATLAB lässt sogar ein wenig Rand außen herum.
Die Farbe wird nach unten automatisch dunkler.



3.6 Die Berechnung Ihrer Ortskurve und Ihrer analytischen Landschaft

Die Ortskurve wird von der Funktion $F_o(p)=FR(p)FS(p)Fr(p)$ des im Abschnitt 3.2 vorgestellten Regelkreises berechnet.

Suchen Sie zuerst den p -Bereich für die Ortskurve, der ein aussagekräftiges Bild ergibt. Dann suchen Sie den K_p -Wert und den T_N -Wert des PID-Reglers, der die Ortskurve etwa durch den Punkt -0.5 auf der reellen Achse laufen lässt. (Parameter T_V setzen Sie immer etwa $T_N/4$).

Eine Erhöhung des K_p -Werts macht den PID-Regler aggressiver. Er reagiert stärker auf Störungen. Dasselbe macht eine Erniedrigung des T_N -Werts. Der Integrator im PID-Regler reagiert schneller.

Haben Sie die richtigen PID-Parameter gefunden, dann berechnen Sie die analytische Landschaft zur Funktion $F(p)=1+F_o(p)$ und sichern die Graphik unter dem Namen, der in der Aufgabenstellung angegeben ist.

Aufgabe 4: Kurvenglättung nach mehreren Methoden, Berechnung von Splines


Aufgabenstellung: 2 getrennte Teilaufgaben a) und b)

a) Erzeugen Sie einen Vektor \mathbf{x} mit $n=300$ Werten x_i aus dem Intervall $[0, 10]$. Zu jedem x_i erzeugen Sie ein y_i . Das zugrunde liegende Datenmodell sei

$$y(x) = (A + \sin(x)) \cdot (B + \cos(x + 0.1)) \cdot e^{-K \cdot x}.$$

Diese „Messwerte“ glätten Sie mit gleitendem Mittelwert und mit exponentieller Glättung.

b) Berechnen Sie mit Splines den Dollbordverlauf eines 21-Meter-Kutters auf der Backbord- und auf der Steuerbordseite. Die Spantenkoordinaten werden gegeben.

Gruppe 3-13 Rechnen Sie mit $A = 1,5$ $B = 1,2$ $K = 0,10$ Exp. Glättung mit $C = 0,7$	Gruppe 14-25 Rechnen Sie mit $A = 2,5$ $B = 0,2$ $K = 0,15$ Exp. Glättung mit $C = 0,8$	Gruppe 1, 2, 26-36 Rechnen Sie mit $A = 3,5$ $B = 0,8$ $K = 0,18$ Exp. Glättung mit $C = 0,9$	Zusatzaufgabe (Dreiergruppe)  Legen Sie selbst die unteren und oberen x-y-Koordinaten fest und verbinden Sie sie durch zwei Spline-Kurven.
Ergebnisse: Die beiden Textfiles <i>Aufgabe4a.txt</i> und <i>Aufgabe4b.txt</i> zur Aufgabe und die beiden Graphiken <i>Aufgabe4Glaettung.tif</i> und <i>Aufgabe4Splines.tif</i> . Für die Zusatzaufgabe: <i>Aufgabe4Fluegelprofil.txt</i> und die Graphik <i>Aufgabe4Fluegelprofil.tif</i>			

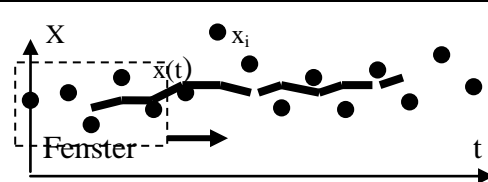
4.1 Kurze Einführung in die Kurvenglättung

Bei der Kurvenglättung verfolgt man mehrere Ziele:

- 1 Minimierung der stochastischen Schwankungen, um ein besseres Bild des Zusammenhangs zu erhalten. Oft müssen die Schwankungen auch beseitigt oder minimiert werden, wenn es sich um Steuersignale für einen Prozess handelt. Zu starke zufällige Schwankungen könnten die Regelung stören. Man benutzt z.B. *gleitende Mittelwerte*, *gleitende Polynome*, *exponentielle Glättung*.
- 2 Suche eines Modells, das die Daten beschreibt. Ideal wäre ein Modell für den gesamten Kurvenverlauf und nicht nur für kleine Stücke. Dem Anwender stehen hier viele mathematische Funktionen zur Verfügung, die man einzeln oder in Kombination den Daten anpassen kann. Hilfsmittel sind Polynome, Fourierreihen, Linearkombinationen von Gaußkurven oder von e-Kurven, ...
- 3 Ein weitergehendes Modell möchte nicht nur die Daten beschreiben, sondern den Prozess beschreiben, der die Messdaten liefert. Dazu müssen die Prozessparameter erkannt und irgendwie in das Modell eingebaut werden. Diese Aufgabe ist die schwierigste. Oft lässt sich der Prozess nur durch ein System gekoppelter Differentialgleichungen beschreiben, was den Rahmen dieser einfachen Aufgabe verlässt.

Glättung durch gleitende Mittelwerte

Man greift sich aus den n Messdaten x_i im Vektor X eine ungerade Anzahl von p aufeinander folgenden Werten heraus (Datenfenster) und berechnet deren Mittelwert. Dann verschiebt man das Datenfenster um eine



Position und berechnet wieder den Mittelwert usw. Es entsteht eine Folge von Mittelwerten, die eine glattere Kurve bilden, als die Originalwerte. Das Verfahren hat (wie jedes Verfahren) auch Nachteile:

- Die geglättete Kurve hat Knickstellen an jedem Knoten.
- Starke Maxima und Minima werden etwas flacher (werden ein wenig gebügelt).
- Es gibt eine Startlücke und eine Endlücke (Zeitverzögerung bei online-Glättung).

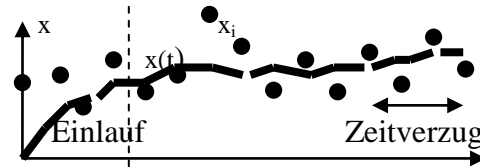
Exponentielle Glättung

Diese Form der Glättung verwendet man vorzugsweise bei der Glättung laufender Prozessdaten, die in endloser Folge von der Messapparatur kommen und an die Steuerung weitergeleitet werden (online-Glättung). Die Glättungsformel ist einfach zu realisieren und arbeitet sehr schnell, hat aber ebenfalls einige Nachteile.

Die Glättungsformel lautet:

$$x(t_i) = C \cdot x(t_{i-1}) + (1-C) \cdot x_i$$

Dabei ist C eine Glättungskonstante $0,5 < C < 1$,
 $x(t_i)$ der geglättete Wert zum Zeitpunkt t_i .
 x_i der aktuelle Messwert zur Zeit t_i .



Arbeitet man z.B. mit dem üblichen Wert $C=0.9$, dann „vergisst“ der Glättungsalgorithmus 10% des geglätteten Wertes vom Zeitpunkt davor und ersetzt den Verlust durch 10% des aktuellen Messwerts x_i . Das Verfahren hat (wie jedes Verfahren) auch Nachteile:

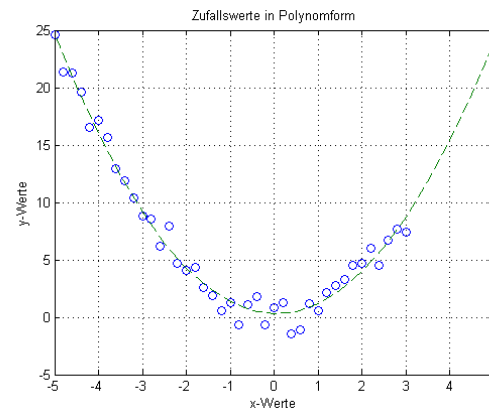
- Die geglättete Kurve hat Knickstellen an jedem Knoten.
- Starke Maxima und Minima werden flacher (werden gebügelt).
- Es gibt eine Einlaufkurve und einen Zeitverzug von $1/(1-C)$ Zeitintervallen Δt . Bei $C=0.9$ sind das 10 Zeitintervalle Δt .

4.2 Herstellung stochastisch gestörter „Messdaten“ und polynomialer Fit

Wir wollen unsere Daten nicht wirklich messen, sondern simulieren. Wir geben uns einen bekannten Kurvenverlauf vor und addieren Zufallszahlen geeigneter Größe. Es entsteht ein gestörter Kurvenverlauf. Lesen Sie die MATLAB-Hilfe zu `randn`.

<pre>x=[-5 : 0.2 : 3]; y=x.^2+randn(size(x)); % Polynomialer Fit p=polyfit(x, y, 2); %Polynom an Stuetzstellen berechnen xx=[-5 : 0.5 : 5]; q=polyval(p, xx); plot(x, y, 'o', xx, q, '--'); xlabel('x-Werte'); ylabel('y-Werte'); title('Zufallswerte in Polynomform'); grid;</pre>	<p>Die x-Koordinaten der „Messpunkte“. Die y-Koordinaten entstehen aus einer Parabel mit addierten zufälligen Fehlern.</p> <p>Ausgleichskurve berechnen (Parabel)</p> <p>Die x-Koordinaten der Stützstellen. Polynomwerte an den Stützstellen xx.</p> <p>Zeichne Messpunkte als Kreise, das Polynom gestrichelt. x-Achse beschriften. y-Achse beschriften. Titel der Graphik. Gitternetz.</p>
--	---

Die Ausgleichsparabel wird nach der Methode der kleinsten Abweichungsquadrate berechnet. Da wir wussten, dass eine Parabel den Messwerten zugrunde liegt, war die Lösung hier sehr einfach. Bei einer realen Messaufgabe kennt man selten das Datenmodell und muss verschiedene Modelle probieren. Aber selbst wenn ein Datenmodell die Daten perfekt wiedergibt, muss es nicht das wahre Prozessmodell sein, sondern bleibt eine mathematische Beschreibung der Daten.



4.3 Glättung mit gleitendem Mittelwert und exponentielle Glättung

Aufgabenstellung: Erzeugen Sie einen Vektor y mit $n=300$ zufällig gestörten Messwerten mit $0 \leq x \leq 10$. Das zugrunde liegende Datenmodell sei nicht wie im Beispiel x^3 , sondern

$$y(x) = (A + \sin(x)) \cdot (B + \cos(x + 0.1)) \cdot e^{-K \cdot x} \quad (A, B, K \text{ siehe Aufgabenstellung}).$$

Diese „Messwerte“ glätten Sie mit gleitendem Mittelwert und mit exponentieller Glättung (mit Ihrem C-Wert aus der Aufgabenstellung), und plotten die beiden Kurven in einem Plot.

Beispielprogramm:

```
% Glättung
```

```
x=[-5 : 0.05 : 5];
```

```
y=0.1*x.^3+randn(size(x));
```

```
% Beachte elementweise Operation .^
```

```
p=polyfit(x, y, 3);
```

```
q=polyval(p, x);
```

```
n=length(x);
```

```
C=0.9;
```

```
yexp=y;
```

```
for i =2 : n;
```

```
yexp(i)=C*yexp(i-1)+(1-C)*y(i);
```

```
end;
```

```
fb=11;
```

```
fm=(fb-1)/2;
```

```
yglm=y;
```

```
for i=1+fm : n-fm;
```

```
yglm(i) = mean( y(i-fm : i+fm) );
```

Glättung mit gleitendem Mittelwert und exponentielle Glättung.

Zuerst Herstellung von 201 fehlerbehafteten Datenwerten als $y(x) = 0.1 \cdot x^3 + \text{Zufallszahl}$

Wir passen ein Polynom 3. Grades an. Die Koeffizienten stehen auf Vektor p . Wir berechnen den Verlauf des Polynoms an allen x -Werten. Vektor q enthält die Werte. Wir besorgen uns die Länge von Vektor x .

Exponentielles Glätten: Glättungskonstante C festlegen. Vektor $yexp$ mit unseren „Messwerten“ y belegen.

Schleife ab $i=2$. (Ab $i=1$ verbietet uns die Glättungsformel)

Glättungsformel.

Ende der Schleife und Ende der exponentiellen Glättung.

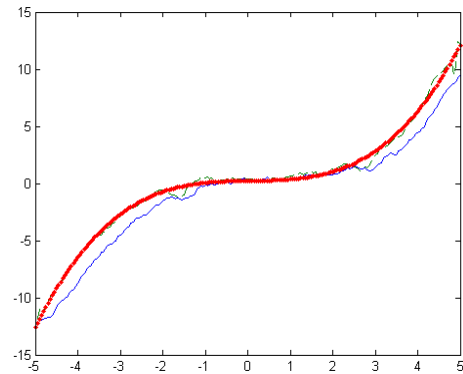
Gleitender Mittelwert: Fensterbreite fb muss eine ungerade Zahl >1 und $\ll n$ sein. Fenstermitte fm (Zählung im Fenster ab 0) Vektor $yglm$ mit unseren „Messwerten“ y belegen.

Schleife mit Anfangs- und End-Lücke von Einer halben Fensterbreite.

Die Fenstermittel mit Funktion *mean* be-

end;	Rechnen und auf Vektor <code>yglm</code> ablegen. Ende der Schleife und Ende der Glättung.
<code>plot(x,yexp,'-', x,yglm,'--',x,q,'')</code>	Ausgabe der drei Kurven

Die dunkle glatte Kurve ist das **Polynom**. Die Glättung mit **gleitendem Mittelwert** versteckt sich in dieser Kurve. Diese Glättung ist demnach sehr effektiv und dem Polynom sehr ähnlich, was darauf zurückzuführen ist, dass wir die Fenstermittel auch tatsächlich dort abgespeichert haben, wo sich die Mitte des Fensters gerade befand. Die **exponentielle Glättung** hat links die kurze Einlaufstrecke. Ansonsten hinkt sie dem Polynom 10 Messwerte hinterher.



Dieses Hinterherhinken liegt daran, dass wir den Zeitverzug graphisch nicht berücksichtigt haben. Bei einigen Anwendungen (z.B. der Auswertung von Börsenkursen) ist dieses Hinterherhinken sogar erwünscht und gibt den Börsianern wichtige Hinweise für den Verkauf oder den Kauf von Wertpapieren.

4.4 Glättung mit Splines

Splines stellen die gesuchte geglättete Kurve stückweise aus Polynomen zusammen. An den Nahtstellen (Knoten) möchte man z.B. Knicke vermeiden (1. Ableitung hat keine Sprünge) oder eine unstetige Änderung der Krümmung (2. Ableitung hat keine Sprünge). Je nach verwendetem Polynomgrad spricht man von quadratischen, kubischen oder allgemein von Splines p -ten Grades. Bei kubischen Splines kann man z.B. verlangen, dass die 1. und die 2. Ableitung der geglätteten Kurve an den Knoten stetig ist.

Splines verwendet man nicht zur Berechnung von Ausgleichskurven, sondern viel mehr um die eleganten Schwünge von gekrümmten Brückenfahrbahnen von Pfeiler zu Pfeiler zu berechnen. Beim Bau von Eisenbahntrassen legen Splines die optimale Verbindungskurve zwischen vorgegebenen Punkten fest. Diese Punkte dürfen nicht zu eng beieinander liegen.

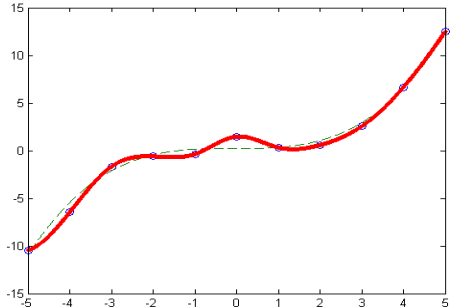
Ihre Spline-Aufgabenstellung: Die 8 Spanten, Bug und Heck eines 21-Meter-Kutters sollen sowohl steuerbords (`ystrb`-Daten) als auch backbords (`ybb`-Daten) durch Splines verbunden werden und damit die Krümmung der Dollbordplanke angeben, d.h. **das Profil des Kutterdecks** von oben gesehen. Plotten Sie beide Kurven, d.h. `ystrb(x)` und `ybb(x)` in einen Plot.

Pos.	Bug	Spa. 1	Spa. 2	Spa. 3	Spa. 4	Spa. 5	Spa. 6	Spa.7	Spa. 8	Heck
<code>x</code>	0	2,333	4,667	7	9,333	11,67	14	16,33	18,67	21
<code>ystrb</code>	0	0,665	1,375	2,021	2,489	2,681	2,526	2	1,131	0
<code>ybb</code>	0	-0,665	-1,375	-2,021	-2,489	-2,681	-2,526	-2	-1,131	0

Beispielprogramm mit Zufallsdaten und nur einer y -Kurve:

<code>% Glättung mit Splines</code>	
<code>x=[-5 : 1 : 5];</code>	
<code>y=0.1*x.^3+0.8*randn(size(x));</code>	Zuerst Herstellung von wenigen fehlerbehafteten Daten.

<pre>p=polyfit(x, y, 3); xx=[-5 :0.01 :5]; q=polyval(p, xx); pp=spline(x,y); v=ppval(pp,xx); plot(x, y, 'o', xx, q,'--', xx, v, '.')</pre>	<p>Wir legen ein Polynom hindurch.</p> <p>Wir erzeugen einen Vektor xx mit 1001 Werten und berechnen dort das Polynom.</p> <p>Berechnung der Spline-Koeffiziententupel. Für jeden Zwischenraum wird ein spezielles Polynom 3. Grades angepasst.</p> <p>Berechnung des Kurvenverlaufs zwischen den Stützstellen.</p> <p>Ausgabe des Plots.</p>
--	---

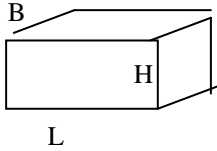
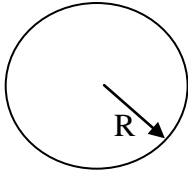
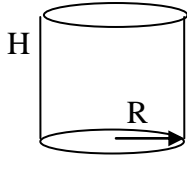
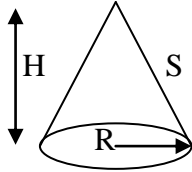
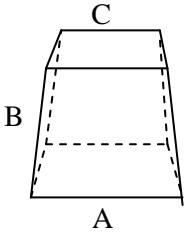
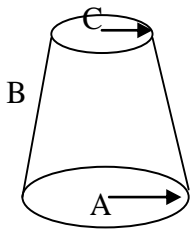
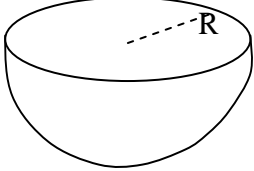
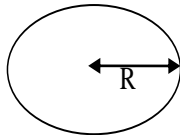
<p>Die 11 Stützstellen sind mit o markiert. Das Polynom ist eine Ausgleichskurve und geht zumeist an den Stützstellen vorbei (Methode der kleinsten Abweichungsquadrate).</p> <p>Die Spline-Kurve geht exakt durch alle Stützstellen und verbindet diese mit eleganten Bögen, die z.B. als Straßenführung oder Eisenbahntrasse geeignet wären.</p>	
--	---

Aufgabe 5: Behälterberechnung mit Daten aus Excel

Aufgabenstellung: Aus einer Exceltabelle sind die Daten von **N** Teilen aus einer Auswahl von 6 verschieden geformten Standardteilen zu lesen. Alle 6 müssen auftreten, einige doppelt. Anschließend werden die Oberfläche, das verbaute Stahlvolumen und das Leergewicht aller Teile berechnet und auf eine Datei ausgegeben.

<p>Gruppe 4-12 Sie arbeiten mit N=10 und den 6 Teilen 1, 2, 3, 5, 6, 7</p>	<p>Gruppe 13-21 Sie arbeiten mit N=9 und den 6 Teilen 2, 3, 4, 6, 7, 8</p>	<p>Gruppe 22-30 Sie arbeiten mit N=11 und den 6 Teilen 1, 2, 3, 4, 7, 8</p>	<p>Gruppe 1-3, 31-36 Sie arbeiten mit N=10 und den 6 Teilen 3, 4, 5, 6, 7, 8</p>
<p>Ergebnisse: Das Textfile <i>Aufgabe5.txt</i> zur Aufgabe, die Exceldatei <i>Aufgabe5.xls</i> und die Ausgabedatei <i>Aufgabe5Ergebnisdaten.txt</i> .</p>			

Die 8 Standardteile mit Typnummer, Maßen und Materialflächen F sind:

1. Quader Maße: L, B, H $F=2LB+2LH+2BH$	2. Kugel Maß: R $F=4\pi R^2$	3. Zylindermantel Maße: R, H $F=2\pi RH$	4. Kegelmantel Maße: R, H $F=\pi RS$ mit $S=\sqrt{R^2+H^2}$
			
5. Pyramidenstumpf Maße: A, B, C $F=A^2+C^2+4bh$ mit $b=(A+C)/2$ und $h=\sqrt{B^2-\left(\frac{A-C}{2}\right)^2}$	6. Kegelstumpfmantel Maße: A, B, C (A>C) $h=\sqrt{B^2-\left(\frac{A-C}{2}\right)^2}$ $S=\sqrt{\left(\frac{hC}{A-C}\right)^2+C^2}$ $F=\pi(A(B+S)-CS)$	7. offene Halbkugel Maß: R $F=2\pi R^2$	8. Kreisscheibe Maß: R $F=\pi R^2$
			

Da wir aus einer Exceldatei keine Texte, sondern nur Zahlen übernehmen können, behelfen wir uns mit der Typnummer. Z.B. der Quader hat Typnummer 1, die Kugel Typnummer 2, usw.

	A	B	C	D	E
1	Teiletyp	Typnummer	1. Mass	2. Mass	3. Mass
2	Kugel	2	0,35		
3	Quader	1	1,4	0,85	0,3
4	Kegelstumpf	6	0,44	1,25	0,67
5	Kugel	2	0,63		
6					

Unsere kleine Exceltabelle heißt X:\C2\Aufgabe5.xls. Die Kugel kommt hier zweimal vor mit unterschiedlichem Radius. Da wir nur Zahlen lesen können, beschränken wir uns hier auf den Bereich B2:E5 der Exceltabelle, den wir auf eine Matrix TD (Teile-Daten) einlesen.

TD=xlsread('X:\C2\Aufgabe5.xls', 'B2:E5')

Teile-Daten aus Exceltabelle lesen.

TD =

Die gelesenen Daten werden

<pre> 2 0.35000 NaN NaN 1 1.40000 0.85000 0.30000 6 0.44000 1.25000 0.67000 2 0.63 NaN NaN mn=size(TD); m=mn(1); Blechdicke=0.008; Rho_Stahl=7800; for i=1:m; typnum=TD(i, 1); switch typnum case 1 % Quader L=TD(i, 2); B=TD(i, 3); H=TD(i, 4); F=2*(L*B+L*H+B*H); case 2 % Kugel R=TD(i, 2); F=4*pi*R^2; otherwise F=NaN; end; TD(i, 5)=F; TD(i, 6)=F*Blechdicke; TD(i, 7)= F*Blechdicke*Rho_Stahl; end; save X:\C2\Aufgabe5Ergebnisdaten.txt TD -ascii </pre>	<p>angezeigt. Leere Zellen aus Excel haben den Wert NaN (not a number). Das Komma der deutschen Excelversion wird richtig als Dezimalkomma interpretiert.</p> <p>Zeilen- und Spaltenzahl von Matrix TD. Zeilenzahl m ist das erste Element im kleinen Vektor <i>mn</i>. Wir legen 8 mm Blechdicke fest. Stahldichte 7800 Kg/m³</p> <p>Wir starten eine Schleife über die m Datenzeilen. Teile-Typnummer steht in Spalte 1. Wir verzweigen mit switch.</p> <p>Fall 1 ist der Quader. Er hat die Maße L, B, und H. Die Fläche wird berechnet.</p> <p>Fall 2 ist die Kugel. Einziges Maß ist der Radius. Die Fläche wird berechnet.</p> <p>Es folgen 4 weitere Fälle für die restlichen 4 Teiletypen (6 aus 8).</p> <p>Bei unbekannter Typnummer setzen wir „not a number“.</p> <p>Ende der switch-Konstruktion.</p> <p>Wir speichern die berechneten Flächen in Matrix TD, 5. Spalte, das Stahlvolumen in die 6. Spalte, die Stahlmasse in die 7. Spalte</p> <p>Ende der for-Schleife.</p> <p>Wir geben die Matrix TD auf eine Textdatei aus.</p>
--	---

Ein Blick auf Datei X:\C2\Aufgabe5Ergebnisdaten.txt (Das E-Format ist ungewohnt zu lesen. Lies „mal 10 hoch“ für das e):

```

2.0000e+000 3.5000e-001 NaN NaN 1.5393e+000 1.2315e-002 9.6057e+001
.....
.....
2.0000e+000 6.3000e-001 NaN NaN 4.9875e+000 3.9900e-002 3.1122e+002

```

Aufgabe 6: Funktionstabellierung mit 2D-Plots

Aufgabenstellung: Machen Sie eine *Parameterdarstellung* der Heuberg-Haeberlein-Funktion. Die Funktion mit den beiden Argumenten η (eta) und γ (gamma) lautet:

$$HH(\eta, \gamma) = \sqrt{\gamma + \sin(\eta)} e^{\gamma + \ln(\eta^2)}$$

<p>Gruppe 5-16 Größe η (eta) soll von -5 bis +5 in Schritten von 0.1 laufen. γ (gamma) ist der Parameter mit den Werten $\gamma = 1,$ $\gamma = 1.25,$ $\gamma = 1.5,$ $\gamma = 1.75$ und $\gamma = 2.$ D.h., 5 Kurven sind in das Koordinatensystem zu zeichnen.</p>	<p>Gruppe 17-28 Größe η (eta) soll von -8 bis +8 in Schritten von 0.2 laufen. γ (gamma) ist der Parameter mit den Werten $\gamma = 0.5,$ $\gamma = 0.75,$ $\gamma = 1.0,$ $\gamma = 1.25$ und $\gamma = 1.5.$ D.h., 5 Kurven sind in das Koordinatensystem zu zeichnen.</p>	<p>Gruppe 1-4, 29-36 Größe η (eta) soll von -2.5 bis +2.5 in Schritten von 0.05 laufen. γ (gamma) ist der Parameter mit den Werten $\gamma = 1,$ $\gamma = 1.5,$ $\gamma = 2,$ und $\gamma = 2.5.$ D.h., 4 Kurven sind in das Koordinatensystem zu zeichnen.</p>	<p>Zusatzaufgabe (Dreiergruppe) Parameterdarstellung der Normalverteilung</p> $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ <p>in einer Graphik mit den beiden Parametern μ und σ. Kurve 1: $\mu=0, \sigma=2$ Kurve 2: $\mu=4, \sigma=2$ Kurve 3: $\mu=2, \sigma=1$ Kurve 4: $\mu=6, \sigma=1$ Verwenden Sie TEX-Codierung von μ und σ für die Beschriftung der Kurven</p>
<p>Ergebnisse: Das Textfile <i>Aufgabe6.txt</i>, die Funktion <i>Aufgabe6Funktion.m</i> und die Graphik <i>Aufgabe6Parameterdarstellung.tif</i>. Die Graphik (ein 2D-Plot) mit Skalenbeschriftung, Gitternetz und Beschriftung der 5 bzw. 4 Einzelkurven im Plot. Für die Zusatzaufgabe: <i>Aufgabe6Zusatz.txt</i> und <i>Aufgabe6Zusatz.tif</i>.</p>			

Es gibt verschiedene Möglichkeiten, eine Funktion zweier Veränderlicher graphisch darzustellen. Die bekanntesten Methoden sind:

- Parameterdarstellung (Kurvenscharen). Die zweite Variable wird zum Parameter
- Höhenliniendarstellung (wie bei einer Landkarte), d.h. ein Blick von oben
- 3-dimensionale Darstellung. Blick schräg auf die „Funktionslandschaft“

Stellen Sie eine Funktionsdefinition der HH-Funktion her und speichern Sie diese als File *HH.m* in Ihren C2-Ordner. Ein Beispiel für die Herstellung einer selbst geschriebenen Funktion finden Sie in Aufgabe 2, Kapitel 2.3.

Die Parameterdarstellung einer Funktion mit mehreren Veränderlicher stellt Kurvenscharen dar. Die erste Kurve ist für den Parameterwert $\gamma = 1$ für alle η -Werte zu zeichnen. Anschließend wird in dasselbe Koordinatensystem die zweite Kurve für den zweiten Parameterwert γ gezeichnet usw.

Stellen Sie den 2D-Plot her und speichern Sie ihn als TIF-Datei unter dem Namen *Aufgabe6Parameterdarstellung.tif* in Ihren C2-Ordner. Das folgende Beispiel zeigt die Herstellung von 2D-Plots. **Für Ihre Aufgabe** benötigen Sie nur eine Abwandlung der letzten 8 Zeilen des Beispiels.

```
% Ein einfaches Beispiel
```

```
y=(-5:0.2:5).^3;
plot(y)
```

```
% Jetzt mit Verschönerungen
```

```
x=-5:0.2:5; y=(x).^3;
plot(x,y);
xlabel('x');
ylabel('y');
title('Graphik y = x ^3');
grid;
% Der Plot kommt sofort. Gibt man danach
% xlabel,...,grid, dann wird das im Hinter-
% grund ausgeführt. Anschließend holt man
% das Bild mit einem Klick auf den Button
% "Figure 1" und macht file -> save as ->
% Name.tif .
```

```
% Jetzt mehrere Kurven im Plott
```

```
x1=0:0.05:8; y1=sin(x1);
x2=0:0.03:4.8; y2=cos(x2);
x3=0:0.02:3.2; y3=sin(x3).*exp(-0.5*x3);
plot(x1,y1,x2,y2,x3,y3);
```

Hätten alle Kurven dieselbe x-Achse, d.h. gingen die x-Werte z.B. für jede Kurve von 0 bis 8, dann müsste man nur einen x-Vektor berechnen. Die Plot-Anweisung würde dann lauten:

```
plot( x,y1, x,y2, x,y3 )
```

Wollen Sie im Plot die einzelnen Kurven beschriften, dann nehmen Sie die Anweisung

```
text ( xPosition, yPosition, Zeichenkette);
```

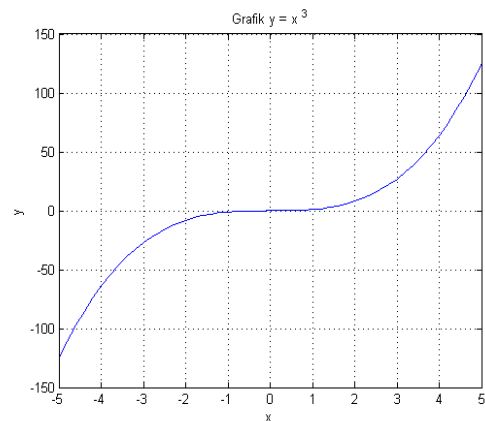
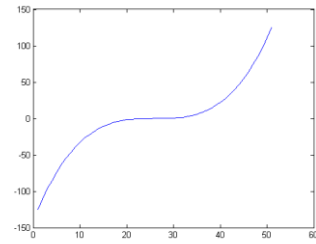
Z.B. `text(3.5, 0.3, 'Sinuskurve');`

Zeichenketten sind in Apostrophe einzuschließen.

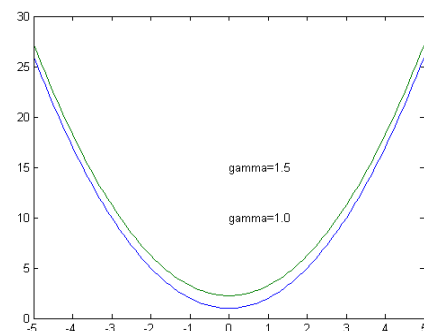
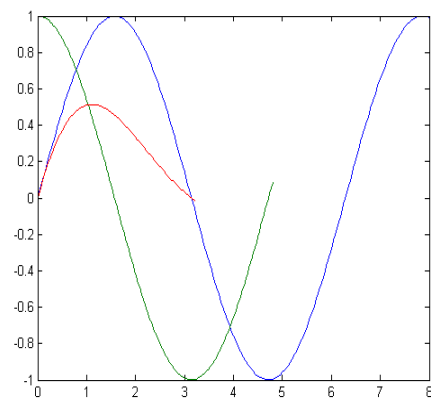
```
% Unsere Beispielfunktion KK mit
```

```
% Beschriftung der Kurven
```

```
x=[-5:0.1:5];
y1=KK(x, 1.0); % Fuer Parameter gamma=1.0
y2=KK(x, 1.5); % Fuer Parameter gamma=1.5
plot(x,y1, x,y2);
text(0, 10, 'gamma=1.0');
text(0, 15, 'gamma=1.5');
```



Wir erzeugen exakt 161 Sinuswerte.
Wir erzeugen exakt 161 Kosinuswerte.
Wir erzeugen exakt 161 Sinus*e-Werte.
Wir plotten die 3 Kurven im selben Koordinatensystem, trotz unterschiedlicher x-Bereiche.



Aufgabe 7: Elastische Membran (Gleichungssysteme lösen)

Aufgabenstellung:

- a) Berechnen Sie die Durchbiegung einer belasteten elastischen nichtsteifen Membran nach einem direkten Verfahren mit 400 inneren Knoten ($N_x=N_y=21$, $F_0=100$ [N] und $ds=1$ [m] und Knoten $K_{10,11}$ mit Last 5 [N]).
- b) Berechnen Sie die Durchbiegung derselben Membran iterativ.
- c) Vergleichen Sie die z-Koordinate des belasteten Knotens aus den Lösungen a) b).
- d) Machen Sie die Rechnung b) nochmals mit $N_x=N_y=201$, $ds=0.1$ und 6 belasteten Knoten. Die Belastungen der einzelnen Knoten in [N] finden Sie unten unter Ihrer Gruppennummer. Speichern Sie die Graphik.

Gruppe 6-14	Gruppe 15-23	Gruppe 24-32	Gruppe 1-5, 33-36
$K_{90,100}$ mit Last 10	$K_{90,100}$ mit Last 30	$K_{90,100}$ mit Last 60	$K_{90,100}$ mit Last 10
$K_{95,90}$ mit Last 20	$K_{95,90}$ mit Last 20	$K_{95,90}$ mit Last 50	$K_{95,90}$ mit Last 60
$K_{105,90}$ mit Last 30	$K_{105,90}$ mit Last 10	$K_{105,90}$ mit Last 30	$K_{105,90}$ mit Last 20
$K_{110,100}$ mit Last 40	$K_{110,100}$ mit Last 60	$K_{110,100}$ mit Last 40	$K_{110,100}$ mit Last 50
$K_{105,110}$ mit Last 50	$K_{105,110}$ mit Last 50	$K_{105,110}$ mit Last 10	$K_{105,110}$ mit Last 30
$K_{95,110}$ mit Last 60	$K_{95,110}$ mit Last 40	$K_{95,110}$ mit Last 20	$K_{95,110}$ mit Last 40

Ergebnisse:

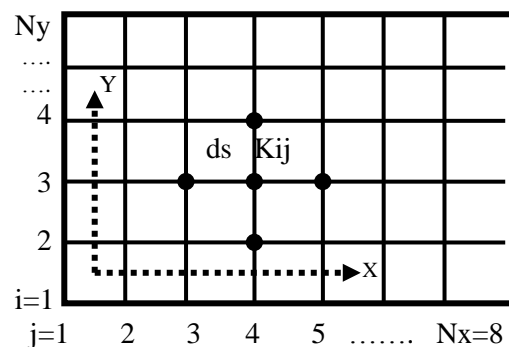
Die 4 Programmtexte *Aufgabe7a.txt*, *Aufgabe7b.txt*, *Aufgabe7d.txt*, ein kleines Textfile mit den 2 z-Werten aus den Aufgaben a) b) mit dem Filenamen *Aufgabe7zWerte.txt* und eine 3-D Graphik *Aufgabe7d.tif*.

7.1 Kleine Einführung zur Membran

Eine elastische Membran ohne eigene Steifheit ist z.B. ein Gummituch, das an den Rändern straff eingespannt ist. Wir können die Membran belasten, z.B. durch eine Kraft in der Mitte oder durch mehrere Kräfte, die im allgemeinen Fall nicht einmal senkrecht zur ungestörten Membran wirken müssen.

Die Membran können wir z.B. durch ein feinmaschiges Netz aus sich kreuzenden, elastischen Fäden modellieren. Die Fäden sind an den Kreuzungsstellen (Knoten) verschweißt. Bereits in der unbelasteten Membran sollen die Fäden eine hohe Vorspannung F_0 [N] haben, d.h. unser Gummituch ist in allen Richtungen gleichmäßig gespannt. Die Vorspannung soll so hoch sein, dass sich die Membran bei Belastung nur wenig eindrückt, etwa wie ein Trampolin. Das vereinfacht die Rechnung ungemein, da wir dann die Veränderung der Fadenspannung durch die Verlängerung beim Durchbiegen in erster Näherung nicht berücksichtigen müssen.

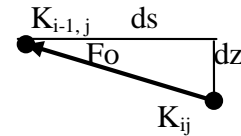
Jeder innere Knoten K_{ij} hat bei einem rechteckigen Maschennetz genau 4 Nachbarn. Die Verbindungsfäden (Strings oder Kanten) zu den Nachbarn sollen alle die gleiche Länge ds mit Spannung F_0 haben. Die Knoten in x-Richtung nummerieren wir mit $j=1, 2, \dots, N_x$, die in y-Richtung mit $i=1, 2, \dots, N_y$. Die vier Nachbarn eines inneren Knotens K_{ij} (kein Randknoten) haben damit die Indizes $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, und $(i, j+1)$.



Wir führen die dritte Koordinate z ein. Die positive z -Achse steht senkrecht auf x und y und zeigt in der obigen Graphik auf den Betrachter. Die z -Koordinate eines Knotens K_{ij} gibt an, wie hoch bzw. wie tief der Knoten eingesunken ist bei einer Durchbiegung der Membran. Randknoten sind unbeweglich und haben alle die Höhe $z=0$.

Ist Knoten K_{ij} um die kleine Strecke dz eingesunken, dann wirkt die Stringspannung F_0 schräg, d.h. sie hat jetzt eine z -Komponente F_z , die den Knoten K_{ij} nach oben und den Knoten $K_{i-1,j}$ nach unten zieht. Der Wert von F_z ist in 1. Näherung

$$F_z = F_0 \cdot dz/ds .$$



Der Knoten K_{ij} ist im Kräftegleichgewicht, wenn alle 4 z -Komponenten der Stringkräfte zu den vier Nachbarn und die Belastung L_{ij} des Knotens K_{ij} sich die Waage halten (zusammen Null ergeben). Wir schauen uns ein Beispiel mit $N_x=6$ und $N_y=5$ an. Für die weitere Berechnung nummerieren wir die 12 inneren Knoten fortlaufend durch. Wir beginnen mit $K_{22}=1$, $K_{32}=2$, usw., d.h. spaltenweise, wie auch MATLAB intern nummeriert.

K_{51}	K_{52}	K_{53}	K_{54}	K_{55}	K_{56}
K_{41}	$K_{42} = 3$	$K_{43} = 6$	$K_{44} = 9$	$K_{45} = 12$	K_{46}
K_{31}	$K_{32} = 2$	$K_{33} = 5$	$K_{34} = 8$	$K_{35} = 11$	K_{36}
K_{21}	$K_{22} = 1$	$K_{23} = 4$	$K_{24} = 7$	$K_{25} = 10$	K_{26}
K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}

Da die Nummerierung „von Hand“ nicht computergerecht ist, suchen wir eine Abbildung der beiden Indizes i und j auf die Zahlenfolge $1, 2, 3, \dots, 12$. (Im allgemeinen Fall auf die Zahlenfolge $1, 2, 3, \dots, (N_x-2)(N_y-2)$). Die fortlaufende innere Knotennummer n ist

$$n = (j-2) (N_y-2) + i-1 .$$

Da sich die Knoten bei einer waagrechten Membran mit senkrechter Belastung nicht von ihrer x - y -Position wegbewegen, reicht es, wenn wir nur die z -Koordinaten z_1, z_2, \dots, z_n der $n=(N_x-2)(N_y-2)$ inneren Knoten berechnen. Für jeden inneren Knoten entsteht eine lineare Gleichung. Nehmen wir an, dass nur Knoten K_8 (Knoten K_{34}) mit einer **Last von 2 [N]** in negative z -Richtung wirkend belastet sei, dann lautet die Kräftegleichung für Knoten K_8 :

$$Fz_{5-8} + Fz_{11-8} + Fz_{7-8} + Fz_{9-8} - 2 = 0 .$$

Wir ersetzen F_z durch $F_0 \cdot dz/ds$, weiterhin ersetzen wir dz durch die z -Koordinatendifferenz der beiden beteiligten Knoten (d.h. des Nachbarn und des Knotens in der Mitte).

Anschließend bringen wir die Belastung auf die rechte Seite der Gleichung und klammern den konstanten Faktor F_0/ds aus. Wir erhalten für Knoten K_8 die lineare Gleichung:

$$\frac{F_0}{ds} \left[(z_5 - z_8) + (z_{11} - z_8) + (z_7 - z_8) + (z_9 - z_8) \right] = 2$$

Betrachten wir die Gleichung genauer, dann treten die z -Koordinaten der 4 Nachbarn positiv mit Koeffizient 1 auf, der Knoten in der Mitte negativ mit Koeffizient 4 (vier mal $-z_8$). Wir bringen den Vorfaktor F_0/ds auf die rechte Seite, und erhalten dort $2 \cdot ds/F_0$.

Das Gleichungssystem für alle 12 inneren Knoten K_1, K_2, \dots, K_{12} mit den 12 Unbekannten, d.h., den z -Koordinaten z_1, z_2, \dots, z_{12} lautet dann:

	z ₁	z ₂	z ₃	z ₄	z ₅	z ₆	z ₇	z ₈	z ₉	z ₁₀	z ₁₁	z ₁₂	Last
K ₁	-4	1			1								0
K ₂	1	-4	1			1							0
K ₃		1	-4	1			1						0
K ₄			1	-4				1					0
K ₅	1				-4	1			1				0
K ₆		1			1	-4	1			1			0
K ₇			1			1	-4	1			1		2 ds/Fo
K ₈				1			1	-4				1	0
K ₉					1				-4	1			0
K ₁₀						1			1	-4	1		0
K ₁₁							1			1	-4	1	0
K ₁₂								1			11	-4	0

Alle nicht besetzten, grau unterlegten Zellen sind Null. Bei Knoten, die einen Randknoten als Nachbarn haben, fehlt ein 1-Koeffizient in der Zeile, bei Knoten, die zwei Randknoten als Nachbarn haben, fehlen zwei 1-Koeffizienten in der Zeile, da ja die z-Koordinate von Randknoten von uns auf z=0 gesetzt wurde. Es entsteht eine sogenannte Bandmatrix **A**, die nur in der Nähe der Diagonalen Werte ungleich 0 hat.

Wir legen ds und Fo fest, z.B. ds=1 und Fo=100 [N]. Damit haben wir ein Gleichungssystem

$$\mathbf{A} \vec{z} = \vec{L}$$

Mit der Koeffizientenmatrix A, dem Lösungsvektor z und der rechten Seite, dem Lastvektor L. Bei einem Maschennetz mit beispielsweise Nx=22 und Ny=22 liegen insgesamt 22x22 Knoten vor, davon 20x20 innere Knoten. Das führt auf ein Gleichungssystem von 400 linearen Gleichungen mit 400 Unbekannten (den gesuchten z-Koordinaten der inneren Knoten).

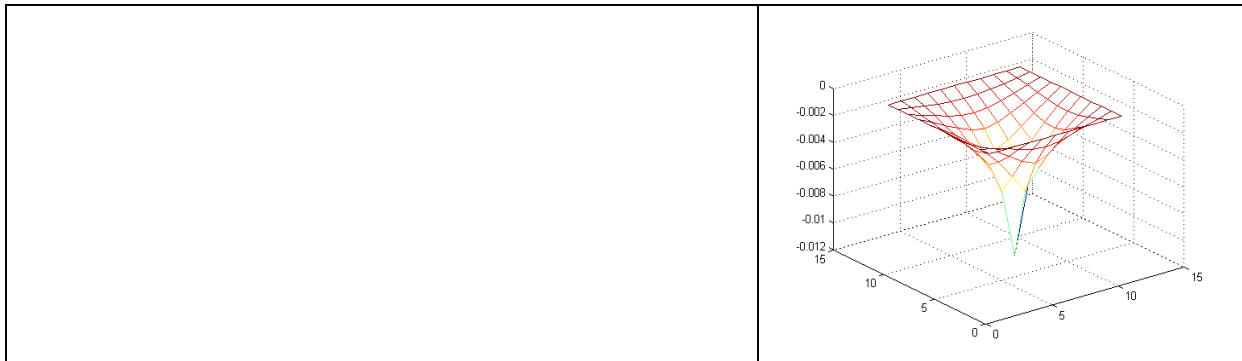
Die numerische Lösung nach einem **direkten Verfahren**, z.B. Gauß-Elimination oder Gauß-Jordan-Verfahren erfordert die Bereitstellung der vollen Koeffizientenmatrix A, im Beispiel eine 400x400 Matrix. Irgendwann geht unserem PC jedoch der Speicherplatz aus (zumal Sie mit einer Studentenversion von MATLAB arbeiten). Im Falle von **Bandmatrizen** könnte man auf eine spezielle Mimik ausweichen, bei der die Bandmatrizen Speicherplatz schonend abgespeichert werden. Diesen Weg wollen wir nicht beschreiten. Bei sehr großen Gleichungssystemen zieht man eine **iterative Lösung** vor. Bei dieser Methode wird eine Anfangslösung (z.B. alle z=0) schrittweise verbessert. Da bei unserem Problem die Diagonale der Koeffizientenmatrix immer Gewicht -4 hat, die restlichen Koeffizienten in der Zeile aber nur 0 oder 1 sind, ist eine iterative Lösung sogar sinnvoll, da sie schnell konvergiert, d.h. rasch auf die richtige Lösung führt.

7.2 Beispielprogramm für eine Lösung mit einem direkten Verfahren

Beispielprogramm für eine direkte Lösung:

ds=1; Fo=100; Nx=11; Ny=11; Last=2; Li=5; Lj=6;	Festlegung der Konstanten
[x, y]=meshgrid(ds:ds:Nx*ds , ds:ds:Ny*ds); % Dimension der Matrizen: Ny Zeilen, Nx Spalten	Herstellung der x-y-Koordinaten-Matrizen für die Graphik

<pre> N=(Nx-2)*(Ny-2); A=zeros(N,N); for ki=1:N; A(ki, ki)= -4; end; % Herstellung Matrix A geht weiter ki=1; % Wir durchlaufen alle N inneren Knoten for j=2:Nx-1; for i=2:Ny-1; % Wir betrachten inneren Knoten Kij mit fortlaufender % Knotennummer n = (j-2)(Ny-2)+i-1 % Randknoten als Nachbarn werden ausgeblendet if j>2; n=(j-2-1)*(Ny-2)+i-1; A(ki, n)=1; end; if j<(Nx-1); n=(j-2+1)*(Ny-2)+i-1; A(ki, n)=1; end; if i>2; n=(j-2)*(Ny-2)+i-1-1; A(ki, n)=1; end; if i<(Ny-1); n=(j-2)*(Ny-2)+i-1+1; A(ki, n)=1; end; ki=ki+1; end; end; % Matrix A ist hergestellt L=zeros(N,1); n=(Lj-2)*(Ny-2)+Li-1; L(n)=Last*ds/Fo; z=linsolve(A,L); zMatrix=zeros(Ny,Nx); ki=1; for j=2:Nx-1; for i=2:Ny-1; zMatrix(i,j)=z(ki); ki=ki+1; end; end; mesh(x,y,zMatrix); % Anzeige der z-Koordinate des belasteten Knotens % Index n ist noch vom L-Vektor richtig gesetzt: z(n) ans = -0.0104 </pre>	<p>Zeilen- bzw. Spaltenzahl N der (N,N)-Koeffizientenmatrix A.</p> <p>Jetzt die gesamte Koeffizientenmatrix A mit Nullen belegen. Die Diagonale von Matrix A mit -4 belegen.</p> <p>Index $ki=1,2,\dots,N$ indiziert die N inneren Knoten Kij.</p> <p>Die Indizes i und j laufen von $j=2,\dots,Nx-1$ und $i=2,\dots,Ny-1$.</p> <p>Die 4 Nachbarn von $K(i,j)$ sind: linker Nachbar $K(i, j-1)$. rechter Nachbar $K(i, j+1)$. Nachbar darunter $K(i-1, j)$. Nachbar darüber $K(i+1, j)$.</p> <p>Nächsten Knoten indizieren. Ende der i-Schleife. Ende der j-Schleife.</p> <p>Herstellung des Spaltenvektors L Knoten $K(Li, Lj)$ wird belastet. Belastung in Vektor L eintragen.</p> <p>Lösung des Gleichungssystems $A z = L$</p> <p>Ausgabe der Lösung als Graphik: Dazu muss aus dem z-Vektor wieder eine Matrix gemacht werden. Immer Ny Werte des z-Vektors ergeben eine innere Spalte von $zMatrix$. Die Ränder sind 0.</p>
---	--



7.3 Beispielprogramm für eine iterative Lösung des Gleichungssystems

Bei der iterativen Lösung benötigen wir speziell in diesem Fall der elastischen, nicht steifen Membran überhaupt keine Koeffizientenmatrix \mathbf{A} , sondern nur die Auslenkungsmatrix \mathbf{z} und die Lastmatrix \mathbf{L} . Zu jedem inneren Knoten $1,2,3,\dots,N$ werden die 4 Nachbarn gesucht, ihre z -Koordinaten addiert und die z -Koordinate des mittleren Knotens 4 mal subtrahiert. Mit Faktor F_0/ds erhalten wir die Resultierende Fz_{ij} der Stringkräfte in z -Richtung. Davon wird die Last L_{ij} subtrahiert, da sie in diesem Membranbeispiel immer nach unten wirkt in negative z -Richtung:

$$Fz_{i,j} = \frac{F_0}{ds} (z_{i,j-1} + z_{i,j+1} + z_{i-1,j} + z_{i+1,j} - 4z_{i,j}) - L_{i,j}$$

Ist der Knoten im Kräftegleichgewicht, dann ist $Fz_{ij} = 0$. Wir verändern die z -Koordinate z_{ij} dieses Knotens nicht. Ist $Fz_{ij} > 0$, dann sind die aufwärts gerichteten Kräfte größer. Wir verschieben den Knoten K_{ij} um die Kleinigkeit dz_{ij} in positive z -Richtung, d.h. wir bilden ein neues z_{ij} nach der Formel

$$z_{i,j}^{neu} = z_{i,j}^{alt} + dz_{i,j}.$$

Ist $Fz_{ij} < 0$, dann sind die abwärts gerichteten Kräfte größer. Wir verschieben den Knoten K_{ij} um die Kleinigkeit dz_{ij} in negative z -Richtung, d.h. wir subtrahieren dz_{ij} .

Ein schwieriges Problem der Iterationsmethodik ist die Festlegung der Verrückung dz_{ij} aufgrund der Kraftkomponente Fz_{ij} . Ein einfaches Gleichsetzen ist sinnlos, da Fz_{ij} eine Kraft in [N] ist, dz aber eine Strecke in [m]. Wir benötigen einen sinnvollen Proportionalitätsfaktor K_p . Damit wird

$$dz_{i,j} = K_p Fz_{i,j}.$$

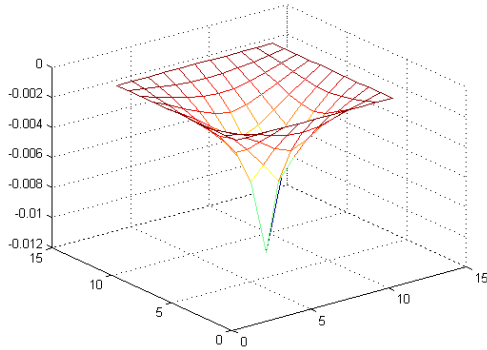
Bei der zahlenmäßigen Festlegung von K_p müssen wir vorsichtig agieren. Ein zu groß gewähltes K_p lässt die Iteration divergieren, d.h. eine sinnlose Lösung berechnen. Ein zu klein gewähltes K_p verschwendet Rechenzeit. Wir lösen dieses Problem durch probieren. Z.B. $K_p = 0.001$ scheint zu funktionieren.

Auch die Anzahl der Iterationen legen wir durch Versuche fest. Wir beginnen mit 10.000 Iterationen und setzen die Zahl dann so weit herab, bis sich unser Ergebnis anfängt zu verändern. Dann gehen wir einen Schritt zurück. Die so gefundene Iterationszahl ist $N_{\text{iterat}} = 200$. Bei größeren Beispielen benötigt man mehr Iterationen.

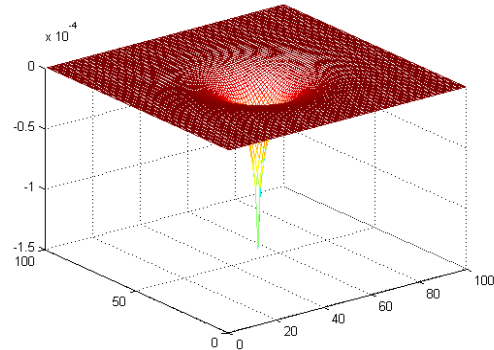
Beispielprogramm für eine iterative Lösung:

<pre> ds=1; Fo=100; Nx=11; Ny=11; Kp=0.001; Last=2; Li=5; Lj=6; N_iterat=200; Fz=0; [x, y]=meshgrid(ds:ds:Nx*ds , ds:ds:Ny*ds); z=zeros(Nx, Ny); L=z; L(Li,Lj)=Last; for iterat=1:N_iterat; for j=2:Nx-1; for i=2:Ny-1; Fz=0; Fz=Fz+z(i, j-1); Fz=Fz+z(i,j+1); Fz=Fz+z(i+1,j); Fz=Fz+z(i-1,j); Fz=(Fz-4*z(i, j))*Fo/ds - L(i, j); dz=Kp*Fz; z(i, j)=z(i, j)+dz; end; end; end; mesh(x,y,z); % Anzeige der z-Koordinate des belasteten Knotens z(Li,Lj) ans = -0.0104 </pre>	<p>Festlegung der Konstanten. Festlegung der Last.</p> <p>Deklaration von Variablen, die in den Schleifen benötigt werden.</p> <p>Herstellung der x-y-Koordinaten-Matrizen für die spätere Graphik.</p> <p>Auslenkungsmatrix $z=0$.</p> <p>Herstellung Lastmatrix $L=0$. Knoten $K(Li, Lj)$ wird belastet.</p> <p>Start der Iterationsschleife.</p> <p>Jeden inneren Knoten behandeln: Die Indizes i und j laufen von $j=2, \dots, Nx-1$, $i=2, \dots, Ny-1$.</p> <p>Startwert der Kräftesumme Fz für den Knoten. Beitrag linker Nachbar Beitrag rechter Nachbar Beitrag Nachbar darunter Beitrag Nachbar darüber</p> <p>Umrechnung der z-Differenzen in Newton und Beitrag der Last.</p> <p>Änderung dz berechnen. Neue z-Koordinate bestimmen.</p> <p>Ende der j-Schleife Ende der i-Schleife</p> <p>Ende der Iterationsschleife.</p> <p>Ausgabe der Lösung als Graphik: Umwandlung des z-Vektors in eine Matrix.</p>
---	---

Die iterative Lösung bringt dasselbe Ergebnis, wie die direkte Lösung des Gleichungssystems, hat aber den Vorteil, mit deutlich kleineren Matrizen zu arbeiten. Wir können größere Membrangitter berechnen.



Mit einem 100x100-Gitter müsste Ihre **Aufgabe d)** eine ähnliche Form haben, wie die Abbildung unten, nur dass 6 Trichter entstehen, da Sie sechs Knoten unterschiedlich belasten.



Aufgabe 8: Differenzialgleichungssystem gekoppelte Pendel

Aufgabenstellung:

- Lösen Sie numerisch das unten gegebene DGL-System der gekoppelten Pendel mit 2 Differentialgleichungen 2-ter Ordnung, und lassen Sie sich den Kurvenverlauf von $x_1(t)$ und $x_2(t)$ in einer Graphik anzeigen.
- Machen Sie dasselbe, wie unter a), nur dass Sie die für die Motorkreisfrequenz ω_M nicht den Wert von ω_0 nehmen, sondern $\omega_M = 0.9 \omega_0$.

Gruppe 7-15

$K_p=1.5$;
 $L_1=1.50$; $M_1=2.0$;
 $D_1=0.08$;
 $L_2=1.51$; $M_2=2.1$;
 $D_2=0.08$;

Gruppe 16-24

$K_p=1.5$;
 $L_1=1.51$; $M_1=2.1$;
 $D_1=0.07$;
 $L_2=1.49$; $M_2=1.9$;
 $D_2=0.08$;

Gruppe 25-33

$K_p=1.5$;
 $L_1=1.54$; $M_1=2.0$;
 $D_1=0.08$;
 $L_2=1.46$; $M_2=2.0$;
 $D_2=0.06$;

Gruppe 1-6, 34-36

$K_p=1.5$;
 $L_1=1.50$; $M_1=2.3$;
 $D_1=0.06$;
 $L_2=1.51$; $M_2=1.7$;
 $D_2=0.08$;

Ergebnisse: Das Textfile *Aufgabe8.txt* und die beiden Graphiken *Aufgabe8aPendel.tif* und *Aufgabe8bPendel.tif*.

8.1 Numerische Lösung von Systemen gewöhnlicher Differenzialgleichungen (Systems of Ordinary Differential Equations – ODE)

Für die numerische Lösung von Differentialgleichungen und Systemen gekoppelter Differentialgleichungen gibt es mehrere Verfahren. Am bekanntesten sind das Euler-Cauchy-Verfahren und das Runge-Kutta-Verfahren. Die beiden Verfahren unterscheiden sich deutlich in Geschwindigkeit und Genauigkeit. Das Runge-Kutta-Verfahren ist in beiden Leistungsmerkmalen besser. Lediglich die Programmierung des Runge-Kutta-Verfahrens ist aufwendiger, als die des Euler-Cauchy-Verfahrens. Da wir jedoch mit fertiger Software (Ode23 bzw Ode45) arbeiten, entfällt dieser Gesichtspunkt. Einziger Grund, sich hier mit dem Euler-Cauchy-Verfahren zu befassen, ist das leichtere Verständnis. Die numerische Lösung von gewöhnlichen Differenzialgleichungssystemen mit dem Euler-Verfahren wird deshalb kurz erläutert (wobei das Runge-Kutta-Verfahren ähnlich arbeitet):

Beispiel: Eine Abkühlkurve nach dem Wärmeleitungsmodell folgt der gewöhnlichen DGL mit konstanten Koeffizienten:

$$T \cdot \dot{x}(t) + x(t) = x_B$$

[s] [K/s] [K] [K]

Die Abkühlung pro Zeiteinheit in [K/s] ist bei reiner Wärmeleitung proportional zur Temperaturdifferenz $x_B - x(t)$. Die Zeitkonstante der Abkühlung sei $T=25$ [s], der Beharrungswert, gegen den die Temperatur strebt, sei 20 [°C]. Die Starttemperatur sei $x_0=90$ [°C]. Die algebraische Lösung dieser einfachen DGL ist bekannt. Es gibt zwei gleichwertige und ineinander überführbare Lösungen:

$$(1) \quad x(t) = x_B + (x_0 - x_B) e^{-t/T}$$

$$(2) \quad x(t) = x_0 + (x_B - x_0) (1 - e^{-t/T})$$

Die numerische Lösung, z.B. nach dem Euler-Verfahren, liefert keine Formel, sondern aufeinanderfolgende t - x -Wertepaare, die graphisch dargestellt die Lösungskurve bilden.

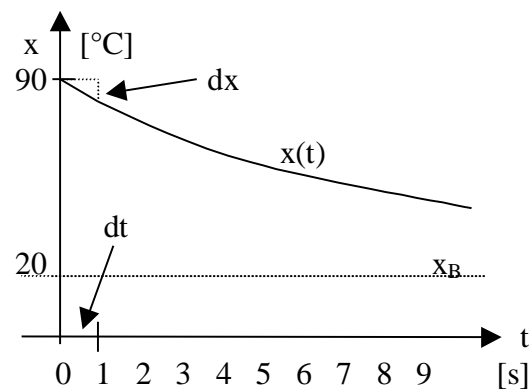
Das Euler-Verfahren: Wir formen die DGL um in $\dot{x} = (x_B - x)/T$ oder $dx/dt = (x_B - x)/T$ oder $dx = ((x_B - x)/T) \cdot dt$. Für einen Zeitschritt wählen wir $dt=1$ [s]. Wir starten mit $x = x_0 = 90$ °C und erhalten

$$(1) \quad dx = ((20 - 90)/25) \cdot 1 = -2,80 \text{ [K]}.$$

Wenn wir in der Zeit um einen Schritt $dt=1$ fortschreiten, sinkt die Temperatur um $dx = -2,80$ Grad. Der neue x -Wert ist dann

$$(2) \quad x_{\text{neu}} = x_{\text{alt}} + dx = 90 - 2,80 = 87,2 \text{ °C}$$

Jetzt setzen wir x_{neu} statt 90 in Gl (1) ein und errechnen das nächste dx und das nächste x_{neu} usw.



Das Runge-Kutta-Verfahren setzt bei der größten Fehlerquelle des Eulerverfahrens an: Euler berechnet den Anstieg der Kurve am Anfang des Zeitintervalls dt . Am Ende des Intervalls hat aber der Anstieg \dot{x} schon einen anderen Wert, d.h. das kleine Kurvenstück trifft sein Ziel am anderen Ende des Intervalls nur ungenau. Viel genauer wäre es, wenn man den Anstieg aus der Mitte des Zeitintervalls nehmen würde. Diese Idee mehrmals verfeinert macht den Hauptteil des Runge-Kutta-Verfahrens aus.

8.2 Aufbereitung von Systemen gewöhnlicher Differenzialgleichungen

MATLAB hat die beiden Funktionen **ode23** und **ode45** zur numerischen Lösung von Differenzialgleichungssystemen. Sie unterscheiden sich in der Ordnung der Approximation. Bei unseren einfachen Systemen ist es egal, welche der beiden Funktionen wir nehmen. Der Aufruf hat folgende Form:

$$[T, Y] = \text{ode45}(\text{'YStrich'}, T_Intervall, Y0);$$

YStrich ist der (eigentlich frei wählbare) Name einer Funktion, die wir selbst schreiben müssen. Beim Aufruf von **ode45** muss der Name der Funktion jedoch in Apostrophen stehen. Die Funktion beschreibt Ihr DGL-System. **T_Intervall** ist ein Vektor mit den beiden Elementen Startzeit **t0** und Endwert **tE**. **Y0** ist der Vektor der Startwerte. **Ode45** liefert als

Resultat einen Vektor mit Zeitwerten (Vektor **T**) und eine Matrix **Y** mit den verschiedenen Lösungskurven der gekoppelten DGLs. Jede Spalte von Matrix **Y** enthält die Werte für eine Kurve.

Der Programmierer der Funktion **YStrich** muss die einmal gewählte Reihenfolge seiner Lösungsvariablen eisern durchhalten. Die Startwerte folgen exakt derselben Reihenfolge. Im folgenden Beispiel sind die Lösungen $x(t)$, $\dot{x}(t)$ und $y(t)$ gesucht. Die beiden gekoppelten DGL des Beispiels lauten:

$$\begin{aligned}\ddot{x} + \dot{y} - 2\dot{x} &= 3 \\ \dot{y} &= 3x + 5\end{aligned}$$

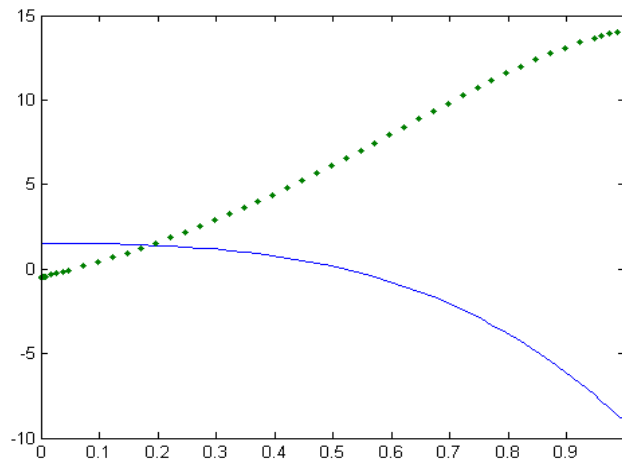
Das System benötigt außer der Startzeit t_0 die 3 Startwerte x_0, \dot{x}_0, y_0 . Damit haben wir schon eine Reihenfolge der Startwerte und damit auch eine Reihenfolge der Funktionswerte festgelegt, die von der Funktion **YStrich** berechnet werden müssen.

Wir schreiben ein Textfile *YStrich.m* und speichern es nach **X:\C2**:

<pre>function dxy= YStrich (t, xy) x = xy(1); xP = xy(2); y = xy(3); xP = xP; yP = 3*x + 2*y + 5; xPP = 3 - yP + 2*xP; dxy = [xP; xPP; yP];</pre>	<p>Unsere Funktion bekommt als Argument den aktuellen t-Wert und einen Vektor xy mit den 3 aktuellen Werten von x, \dot{x} und y übermittelt. Sie soll daraus einen kleinen Vektor dxy berechnen, der die 3 Werte \dot{x}, \ddot{x}, \dot{y} in dieser Reihenfolge enthält (immer einen Punkt mehr, als in xy).</p> <p>Wir extrahieren die Variablen x, \dot{x} und y aus dem Vektor xy.</p> <p>Wir berechnen die 3 Ableitungen laut DGL-System. (Bei x Punkt ist nichts zu berechnen.) Zuerst die niederen Ableitungen xP und yP, dann erst die höheren Ableitungen, hier xPP.</p> <p>Ausgabevektor dxy unbedingt in der richtigen Reihenfolge füllen (wie in xy).</p>
--	--

Das eigentliche MATLAB-Programm ist hier sehr kurz:

<pre>[T , Y] = ode45('YStrich' , [0 1] , [1.5, 0.2, -0.5]); plot(T,Y(:,1),'-', T, Y(:,3),'-');</pre>	<p>x, \dot{x}, y ist die Reihenfolge der Variablen. Wir integrieren von $t_0=0$ bis $t_E=1$ mit Startwerten $x_0=1.5$, $\dot{x}_0=0.2$, $y_0=-0.5$</p> <p>Wir plotten nur die Kurven $x(t)$ und $y(t)$ (Spalte 1 und 3 der Matrix). Die Kurve $\dot{x}(t)$ plotten wir nicht.</p>
---	--



Die Graphik zeigt $x(t)$ durchgezogen, $y(t)$ gepunktet.

Die Umwandlung einer DGL oder eines Systems gekoppelter DGLs als Aufbereitung für eine numerische Lösung soll noch an einem weiteren Beispiel gezeigt werden:

$$x^2 y'' - y'^2 + y'' y = \frac{x}{1+z^2} \quad \text{und} \quad T z' + z y = K$$

mit den Konstanten T und K , und den gesuchten Lösungen $y(x)$ und $z(x)$. Das Gleichungssystem ist ein System von zwei gekoppelten nichtlinearen DGL. x ist die unabhängige Variable.

Als erste Umformung löst man beide DGLs des Systems jeweils nach ihrer höchsten Ableitung auf:

$$y'' = \frac{\frac{x}{1+z^2} + y'^2}{y + x^2} \quad \text{und} \quad z' = \frac{K - z y}{T}$$

Wir schreiben die Funktion *YStrich2.m* und speichern sie nach $X:\C2$

<pre>function Vektoryz= YStrich2 (x, yz) Vektoryz=zeros(3,1); %Spaltenvektor K=7.3; T=25; y = yz(1); yP = yz(2); z = yz(3); yP = yP; zP = (K-z*y)/T; yPP = (x/(1+z^2)+yP^2)/(y+x^2); Vektoryz = [yP; yPP; zP];</pre>	<p>Vektor yz mit den 3 aktuellen Werten von y, \dot{y} und z. Funktion <i>YStrich2</i> soll daraus Vektoryz berechnen, der die 3 Werte \dot{y}, \ddot{y}, \dot{z} enthält.</p> <p>Festlegung der Konstanten Wir extrahieren die Variablen y, \dot{y} und z aus dem Vektor yz.</p> <p>Wir berechnen die 3 Ableitungen laut DGL-System.</p> <p>Ausgabevektor <i>Vektoryz</i> füllen.</p>
---	--

Das eigentliche MATLAB-Programm ist hier sehr kurz:

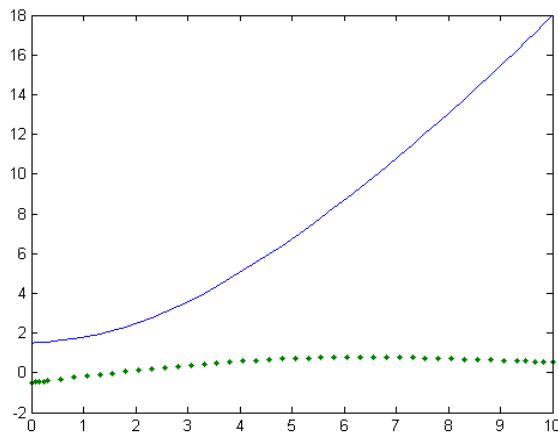
```
[ T , Y ] = ode45( 'YStrich2' , [0 1] , [1.5,
0.2, -0.5] );
```

```
plot(T,Y(:,1),'-', T, Y(:,3),'');
```

y, \dot{y}, z ist die Reihenfolge der Variablen.

Wir integrieren von $t_0=0$ bis $t_E=1$ mit Startwerten $y_0=1.5, \dot{y}_0=0.2, z_0=-0.5$.

Wir plotten nur $y(t)$ und $z(t)$.



Die Graphik zeigt $y(t)$ durchgezogen, $z(t)$ gepunktet

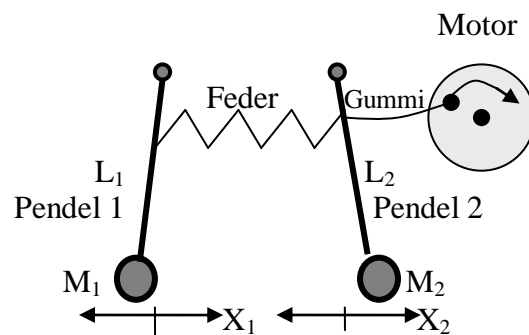
8.3 Ihre Aufgabe: Gekoppelte Pendel

Zwei Pendel mit Masse M und Länge L sind über eine Feder gekoppelt. Die Ausschläge der Pendel sind x_1 und x_2 .

Stößt man ein Pendel an, dann schwingt es.

Über die Feder wird das andere Pendel ebenfalls in Schwingung versetzt,

Die Energie dafür wird dem ersten Pendel entzogen, solange bis es steht. Dann kehrt sich der Vorgang um. Ohne Dämpfung lief das Wechselspiel endlos. Über eine motor-



getriebene Scheibe und ein elastisches Gummiband (an einer Nocke der Scheibe befestigt) wird Pendel 2 ständig mit einer Kraft beaufschlagt, die etwa die Form $K_0 + K_1 \cdot \sin(\omega_M t)$ hat. K_0 ist eine Konstante mit der Dimension einer Kraft, d.h. [N], und hat den Wert der Kraft des Gummibands, wenn die Nocke der sich drehenden Scheibe gerade über oder unter der Nabe ist (Mittelwert der Nockenbewegung). Konstante K_1 hat die gleiche Dimension [N], aber die entsteht aus dem Produkt der Federkonstanten des Gummis mit der Dimension [N/m] und der periodischen Auslenkung x der Nocke in x -Richtung (d.h. waagrecht) mit der Dimension einer Strecke, d.h. [m]. Zusammen ergibt das Produkt *Federkonstante · Auslenkung* also wieder eine Kraft, d.h. [N]. Die Konstante ω_M ist die Kreisfrequenz der Motordrehung mit $\omega_M = 2\pi \cdot f$ mit Motordrehzahl f [U/s].

Die Schwingungs-DGL des Pendels 1 ist für kleine Ausschläge angenähert die eines harmonischen Oszillators:

$$M_1 \ddot{x}_1(t) + D_1 \dot{x}_1(t) + \frac{M_1 g}{L_1} x_1(t) = K_p (x_2(t) - x_1(t))$$

Jeder der vier Terme von ganz links bis ganz rechts hat die Dimension einer Kraft: Trägheitskraft, Bremskraft, Rückstellkraft der Pendelmass, Federkopplungskraft. Dabei ist D_1 [N/(m/s)] eine Dämpfungskonstante und g [m/s²] die Erdbeschleunigung. K_p [N/m] ist die Federkonstante der Kopplungsfeder zwischen den Pendeln.

Die DGL von Pendel 2 ist analog aufgebaut, hat aber auf der rechten Seite den zusätzlichen Störterm $K_0 + K_1 \cdot \sin(\omega_M t)$, der addiert wird. Wählen Sie ein K_0 zwischen 5 und 10 [N] und ein K_1 mit $2 \leq K_1 \leq K_0$. Als Kreisfrequenz ω_M des Motors wählen Sie exakt die der ungedämpften Eigenfrequenz ω_0 von Pendel 2 mit $\omega_0^2 = g/L_2$ mit der Dimension [1/s²] bzw. [Rad/s²], was dasselbe ist, den der Radiant *Rad* ist die Einheit des dimensionslosen Bogenmaßes.

Gesucht sind die 4 Funktionen $x_1(t)$, $\dot{x}_1(t)$, $x_2(t)$, $\dot{x}_2(t)$, wobei uns nur die Schwingungsverläufe $x_1(t)$ und $x_2(t)$ wirklich interessieren. Die beiden anderen Funktionen, $\dot{x}_1(t)$ und $\dot{x}_2(t)$, werden von MATLAB automatisch mitberechnet. Die Aufbereitung der ersten der beiden DGL (Auflösung nach der höchsten Ableitung) ergibt die Gleichung:

$$\ddot{x}_1 = \left[K_p (x_2 - x_1) - D_1 \dot{x}_1 - \frac{M_1 g}{L_1} x_1 + \dots \right] / M_1.$$

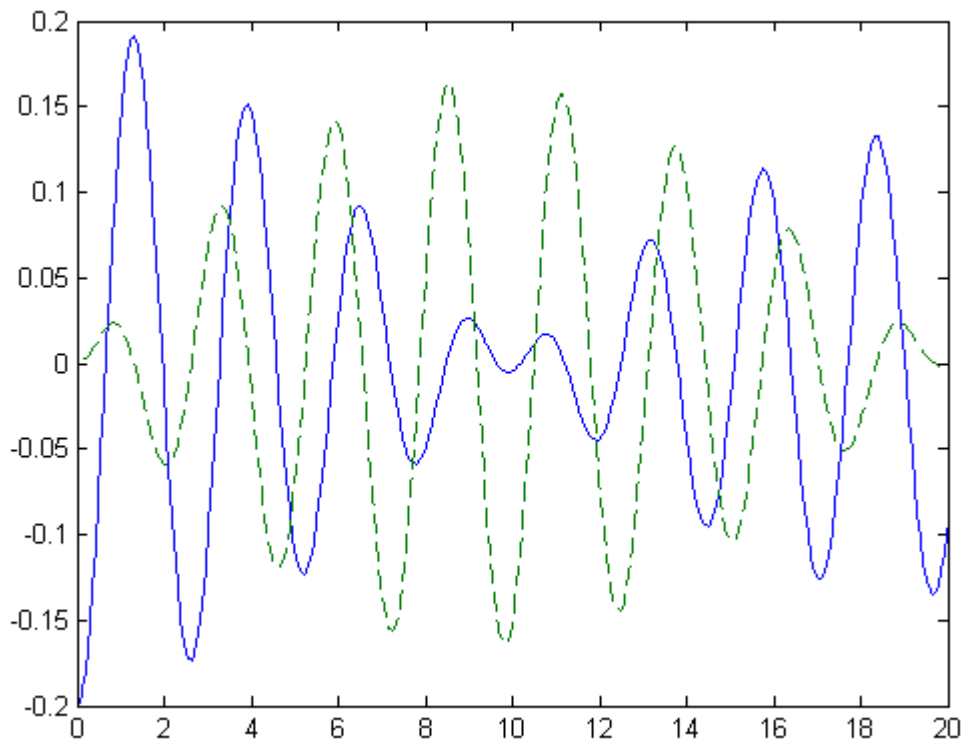
Die zweite Gleichung sieht ähnlich aus, enthält aber zusätzlich noch den Störterm $+K_0 + K_1 \cdot \sin(\omega_M t)$ in der eckigen Klammer. Zu den Zahlenwerten von K_0 , K_1 und ω_M Siehe weiter oben.

Wir schreiben die Funktion *YStrich8.m* und speichern sie nach X:\C2

<pre>function Vektorxx = YStrich8 (t, xx) Vektorxx=zeros(4,1); % Spaltenvektor Kp=1.5; g=9.81; L1=1.50; M1=2.0; D1=0.08; L2=1.51; M2=2.0; D2=0.08; % Hier die Berechnung von omegaM einfügen x1 = xx(1); x2 = xx(2); x1P = xx(3); x2P = xx(4); x1PP = x1P; x2PP = x2P; x1PP = (Kp*(x1-x2)-D1*x1P-M1*g*x1/L1) / M1 ; x2PP = (Kp*(x2-x1)-D2*x2P-M2*g*x2/L2) / M2 ; Vektorxx = [x1P; x2P; x1PP; x2PP] ;</pre>	<p>Vektor xx mit den 4 aktuellen Werten von $x_1, x_2, \dot{x}_1, \dot{x}_2$. Funktion <i>YStrich3</i> soll daraus Vektorxx berechnen, der die 4 Werte $\dot{x}_1, \dot{x}_2, \ddot{x}_1, \ddot{x}_2$ enthält.</p> <p>Festlegung der Konstanten</p> <p>Wir extrahieren die Variablen y, \dot{y} und z aus dem Vektor xx.</p> <p>Wir berechnen die 4 Ableitungen laut DGL-System.</p> <p>Aber noch ohne Störterm!!!</p> <p>Ausgabevektor <i>Vektorxx</i> füllen.</p>
---	--

Das eigentliche MATLAB-Programm zu dieser Aufgabe könnte z.B. sein:

<pre>% gekoppelte Pendel [T, Y] = ode45('YStrich8', [0, 20], [-0.2, 0, 0, 0]); plot(T, Y(:, 1), '-', T, Y(:, 2), '--');</pre>	<p>Wir wollen von $t_0=0$ bis $t_E=20$ integrieren. Startwerte sind $x_{1,0}=-0.2, x_{2,0}=0, \dot{x}_{1,0}=0, \dot{x}_{2,0}=0$, d.h., Pendel 1 ist um 0.2 [m] links ausgelenkt und in Ruhe. Pendel 2 steht im Nullpunkt.</p> <p>Ausgabe der beiden Kurven $x_1(t)$ und $x_2(t)$</p>
---	---



Die Graphik der beiden Schwingungskurven:
 $x_1(t)$ durchgezogen, $x_2(t)$ gestrichelt

Aufgabe 9: Integrodifferenzialgleichungssystem mit Nebenbedingung (PID-Regler)

Aufgabenstellung:

Lösen Sie numerisch das unten gegebene DGL-System der gekoppelten DGL des Regelkreises aus Aufgabe 3 (Graphik *Aufgabe9a.tif*) und suchen Sie durch Verändern der PID-Parameter K_{PR}, T_N, T_V eine Lösung mit minimaler Ausregelzeit (Graphik *Aufgabe9b.tif*).

Gruppe 8-16

$K_{PS}=15$

Gruppe 17-25

$K_{PS}=16$

Gruppe 26-34

$K_{PS}=17$

Gruppe 1-7, 35, 36

$K_{PS}=14$

$T_1 = 28$ Sollwert $W = 800$	$T_1 = 25$ Sollwert $W = 780$	$T_1 = 30$ Sollwert $W = 820$	$T_1 = 7$ Sollwert $W = 850$
Ergebnisse: Die Textfiles <i>Aufgabe9.txt</i> und <i>YSTRICH9.m</i> und die beiden Graphiken <i>Aufgabe9a.tif</i> und <i>Aufgabe9b.tif</i> .			

Eine Integrodifferenzialgleichung ist eine DGL, die neben Ableitungen auch Integrale enthält. Die DGL des sogenannten *idealen PID-Reglers* ist ein Beispiel:

$$y(t) = K_{PR} \left[x_d(t) + \frac{1}{T_N} \int_{u=0}^t x_d(u) du + T_V \dot{x}_d(t) \right]$$

$y(t)$ ist hier der Reglerausgang (Stellgröße), x_d die Regeldifferenz mit $x_d = W - x$ (Sollwert – Istwert). Mit der Stellgröße steuert man gegen eine Regeldifferenz an, d.h. man versucht diese Abweichung vom Sollwert zum verschwinden zu bringen. Bei einer sauberen Regelung ist $W = x$, d.h. $x_d = 0$.

Der Reglerausgang $y(t)$ setzt sich beim PID-Regler aus 3 Bestandteilen zusammen:

- Dem P-Anteil (Proportionalanteil) berechnet sich aus dem Produkt $K_{PR} \cdot x_d(t)$
- Dem I-Anteil (Integrator) berechnet sich aus dem Integral $\int (K_{PR} / T_N) \cdot x_d(t) dt$
- Dem D-Anteil (Differentieller Teil) berechnet sich aus der Ableitung $K_{PR} \cdot T_V \cdot \dot{x}_d(t)$

Die 3 Konstanten K_{PR} , T_N und T_V bedeuten:

K_{PR} oder Reglerverstärkung (mit dem K_{PR} -fachen der Regeldifferenz x_d muss man gegen diese Abweichung vom Sollwert gegensteuern).

T_N oder Nachstellzeit (In der Zeit T_N haben sich die Abweichungen x_d so weit aufintegriert, dass sie denselben Beitrag $K_{PR} \cdot x_d(t)$ liefern, wie in der P-Anteil sofort liefert).

T_V oder Vorhaltezeit (ein Knick oder eine Krümmung in der $x_d(t)$ -Kurve löst einen Steuerimpuls der Stärke $K_{PR} \cdot T_V \cdot \dot{x}_d(t)$ aus, mit dem T_V Tastzeiten lang gegengesteuert wird).

Als Ausregelzeit **T_{aus}** wird die Zeit bezeichnet, ab der die Regeldifferenz $x_d(t)$ nach einem Sollwertsprung ΔW (oder nach einer Störung z_x) die Toleranzgrenzen $\pm Tol$ nicht mehr überschreitet bzw. unterschreitet, d.h. der Zeitpunkt, ab dem $|x_d(t)| < Tol$ ist.

Ihre Aufgabe ist: Lösen Sie das unten gegebene DGL-System, das zusammen einen Regelkreis beschreibt, und lassen Sie sich die Sprungantwort auf einen Sollwertsprung graphisch anzeigen (*Aufgabe9a.tif*). Startwerte sind $x_o = 20^\circ C$, $\dot{x}_o = 0$ [K/s], $y_o = 0$, und $\int x_d = 0$. Sollwert W ist $80^\circ C$. Dann verändern Sie die Reglerparameter K_{PR} , T_N , T_V so, dass eine Ausregelzeit **T_{aus}** <?? Sekunden erreicht wird, und speichern die Graphik (*Aufgabe9b.tif*).

Ofen mit PT_2 -Verhalten $K_{PS} = 16,2$ [K / %] Streckenverstärkung in Grad pro % Ventilhub
 $T_1 = 8,5$ [s] Zeitkonstante eines Verzögerungsgliedes
 y ist Ausgang des Reglers in % Ventilhub

$$DGL: T_1 \ddot{x}(t) + 2T_1 \dot{x}(t) + x(t) = K_{PS} y(t)$$

Messung mit PT_1 -Verhalten $K_p = 1,0$ [K/K] Verstärkungsfaktor

$T = 2,7$ [s] Zeitkonstante der Messverzögerung
 $x(t)$ ist der Ausgang des Ofenmodells PT_2 in °C.

$$\text{DGL: } T \dot{x}_1(t) + x_1(t) = K_p x(t)$$

Idealer PID-Regler

$K_{PR} = 10,0$ [% / K] Reglerverstärkung in % Hub pro Grad
 Abweichung

$T_N = 17,5$ [s] Nachstellzeit des I-Anteils des PID-Reglers

$T_V = 3,5$ [s] Vorhaltezeit des D-Anteils des PID-Reglers

$$\text{DGL: } y(t) = K_{PR} \left[x_d(t) + \frac{1}{T_N} \int_{u=0}^t x_d(u) du + T_V \dot{x}_d(t) \right]$$

mit $x_d = W - x_1$ und $\dot{x}_d = -\dot{x}_1$.

Ihre Aufbereitung der Gleichungen sieht dann für Aufgabe 8a) folgendermaßen aus:

Ofen:	$\ddot{x}(t) = \dots$	Mit den 3 Lösungsfunktionen $x(t)$, $x_1(t)$, $y(t)$
Messung	$\dot{x}_1(t) = \dots$	
PID-Regler	$x_d = \dots$	
	$y(t) = \dots$	

Sie schreiben eine Funktion **YSTRICH9** und speichern sie als **YSTRICH9.m** nach **X:\C2**

function Vektorxxy = YSTRICH9 (t, xxy)	Die Funktion bekommt außer der Zeit t noch den Vektor xy mit den Elementen $[x_d, x, \dot{x}, x_1]$ geliefert und soll daraus <i>Vektorxxy</i> berechnen mit den Elementen $x_d, \dot{x}, \ddot{x}, \dot{x}_1$.
Vektorxxy=zeros(4,1);	Vektorxxy soll ein Spaltenvektor mit 4 Zeilen und einer Spalte sein.
Kps=16.2; T1=28.5; Kp=1; T=0.7; Kpr=10.0; Tn=37.5; Tv=5.5;	Festlegung der Konstanten
global SollwertW T_aus Regeltoleranz;	W, T_{aus} und Tol als globale Variable deklariert
IntegralXd = xxy(1); x = xxy(2); xP = xxy(3); x1 = xxy(4);	Umspeichern der Vektorelemente xy auf Variablen mit Namen, die der Aufgabe angepasst sind.
xd = SollwertW - x1;	Regeldifferenz $x_d = W - x_1$
if abs(xd)>Regeltoleranz; T_aus=t; end;	Ausregelzeit T_{aus} bestimmen.
xP=xP; x1P=(Kp*x-x1)/T;	\dot{x} wird einfach übernommen
y=Kpr*(xd+(1/Tn)*IntegralXd-Tv*x1P);	Die nach \dot{x}_1 aufgelöste Messung-DGL

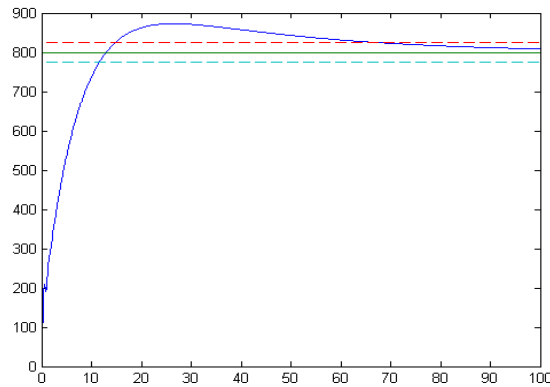
<pre>xPP =(Kps*y-2*T1*xP-x)/T1 ; Vektorxy = [xd; xP; xPP; x1P] ;</pre>	<p>Reglerausgang y, wobei \dot{x}_d durch $-\dot{x}_1$ ersetzt wurde</p> <p>Die nach \ddot{x} aufgelöste Ofen-DGL</p> <p><i>Vektorxy</i> in der richtigen Reihenfolge füllen mit den Elementen $x_d, \dot{x}, \ddot{x}, \dot{x}_1$.</p>
--	--

Im eigentlichen MATLAB-Programm vereinbaren wir dieselben globalen Variablen. Damit ist ein einfacher Austausch von Werten zwischen Funktion und Programm möglich. Nehmen Sie aber als Namen für globale Variable immer lange Namen, damit nicht die Gefahr besteht, dass ein Name schon anderweitig in MATLAB als Variable verwendet wird.

Der Sollwert W soll als durchgezogene waagrechte Linie gezeichnet werden, die Toleranzgrenzen dicht darüber und darunter als gestrichelte waagrechte Linien.

<pre>global SollwertW T_aus Regeltoleranz; SollwertW=800; T_aus=0; Regeltoleranz=25; [T, Y] = ode45('YSTRICH9', [0, 100], [0, 20, 0, 20]); [m,n]=size(Y) ; for i=1 :m ; W(i)=SollwertW; Wto(i)=SollwertW+Regeltoleranz; Wtu(i)=SollwertW-Regeltoleranz; end; plot(T, Y(:, 2),'-', T,W, '-', T,Wto,'--', T,Wtu,'--');</pre> <p>T_aus T_aus = 67.8201</p>	<p>Globale Variable deklarieren und Werte zuweisen.</p> <p>Integrieren von $t_0=0$ bis $t_E=50$. Startwerte sind $\int x_d=0, x_o=20^\circ\text{C}, \dot{x}_o=0, x_{1,o}=20^\circ\text{C}$.</p> <p>Zeilenzahl m und Spaltenzahl n von Y.</p> <p>Füllen von 3 Vektoren mit derselben Länge m wie eine Y-Spalte mit immer demselben Sollwert und seinen Toleranzgrenzen.</p> <p>Plotten der Graphik mit der Sprungantwort $x(t)$ aus Spalte 2 von Y und dem Sollwert mit seinen Toleranzgrenzen.</p> <p>Anzeige der Ausregelzeit T_{aus}.</p>
---	---

Die Graphik der Sprungantwort mit Sollwert und Toleranzgrenzen der Regelung über der Zeit aufgetragen ist:



Aufgabe 10: Partielle DGL – Die ebene Wellengleichung Animation eines Randwertproblems

Aufgabenstellung: Simulieren Sie die Wellenausbreitung von Oberflächenwellen (gespannte Membran mit vorgegebenen Randbedingungen und Punktquellen).

Gruppe 1-9 ds=0.05, Fo=100 Nx=62, Ny=82 M=0.5	Gruppe 10-18 ds=0.03, Fo=200 Nx=62, Ny=82 M=1	Gruppe 19-26 ds=0.025, Fo=100 Nx=52, Ny=102 M=0.5	Gruppe 27-36 ds=0.03, Fo=200 Nx=52, Ny=102 M=1
Ergebnisse: Programmfile Aufgabe10.txt und Vorführung der Animation			

10.1 Partielle Differenzialgleichungen - Wellengleichung

Partielle Differenzialgleichungen enthalten eine abhängige Variable und *mehrere* unabhängige Variable. Als unabhängige Variable tritt bei allen dynamischen Problemen die Zeit t auf und bei den meisten Anwendungen zusätzlich eine oder mehrere Raumkoordinaten. Typische Beispiele sind Diffusionsvorgänge und Wellenausbreitung.

Die Diffusionsgleichung in einer Raumrichtung:
$$\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2}.$$

Hierbei ist c eine Konzentration in $[\text{mol}/\text{m}^3]$, D der Diffusionskoeffizient in $[\text{m}^2/\text{s}]$ und x die Koordinate der Ausbreitungsrichtung in $[\text{m}]$, z.B. bei der Diffusion in einem engen Kanal. Gesucht ist die Funktion $c(x,t)$, die sowohl die partielle DGL erfüllt als auch die Rand- und Startbedingungen.

Wird eine Membran durch eine punktförmige Störung zu einer Schwingung angeregt, dann breiten sich die Störungen in Form ringförmiger Wellen aus. Die ebene Wellengleichung in Polarkoordinaten lautet

$$\frac{1}{r} \frac{\partial^2}{\partial r^2} (r \cdot z(r,t)) = \frac{1}{v^2} \frac{\partial^2}{\partial t^2} (z(r,t)).$$

Dabei ist $z(r, t)$ die Auslenkung der ebenen Membran in die dritte Raumkoordinate z in [m]. Radius r ist der Abstand vom Zentrum in [m] und v ist die Ausbreitungsgeschwindigkeit der Wellen in [m/s]. Die zweite Polarkoordinate, der Winkel φ , tritt hier nicht auf, da ein zentralsymmetrisches Problem vorliegt, d.h., dass jede Richtung φ auf der Ebene gleich behandelt wird (ringförmige Wellen), die vom Zentrum ausgehen.

Je nach Form der Membran (quadratisch, rechteckig, rund) und der Art der Randbedingungen (endlos, fest eingespannt, lose Ränder) erhält man unterschiedliche Lösungen, die z.B. auf Summen von Sinus- und Kosinusfunktionen oder auch auf Besselfunktionen führen.

Wir wollen eine eingespannte elastische Membran untersuchen, die eine Massenbelegung ρ [Kg/m²] und eine Spannung F_0 in [N/m] besitzt. Wir modellieren die Membran wie in Aufgabe 7 durch ein Gitter elastischer Fäden im Abstand ds , die an den Knoten verschweißt sind. Die Massenbelegung erzeugen wir durch eine Massenbelegung m der Knoten. Jeder Knoten hat jetzt die Möglichkeit zu schwingen, wenn er angestoßen wird, da er eine eigene Masse besitzt und von seinen Nachbarn durch elastische Fäden gehalten wird. Wir betrachten nur Anregungen in der z -Richtung. Eine solche Anregung könnte sein:

- Eine Startauslenkung $z_{i,0}$ des i -ten Knotens.
- Eine Startgeschwindigkeit $\dot{z}_{i,0}$ des i -ten Knotens.
- Eine Kombination aus beiden.
- Eine immer wiederkehrende Anregung in Form einer zeitabhängigen Funktion.

Wie bei dem gekoppelten Pendel in Aufgabe 8 wird sich die Energie des Knotens K_i auf seine Nachbarn übertragen. Diese beginnen ebenfalls zu schwingen, regen wiederum ihre Nachbarn an. Die Folge ist eine Ausbreitung der Störung über die gesamte Membran. Ränder reflektieren die einfallenden Wellen.

Die Schwingungs-DGL eines einzelnen Knotens K_i ist beim Fehlen jeglicher Dämpfung und Berücksichtigung der 4 direkten Nachbarknoten

$$m \ddot{z}_i = \frac{F_0}{ds} \sum_{j=1}^4 (z_j - z_i).$$

Links steht die Trägheitskraft, rechts die Summe der vier z -Komponenten der Zugkräfte der vier Nachbarn. Auch hier wird nur von kleinen Auslenkungen z ausgegangen. In diesem Falle ist die Neigung $(z_j - z_i)/ds$ der elastischen Fäden zwischen zwei Knoten klein. Das Fadenstück verlängert sich nur unmerklich und wir können mit einer konstanten Fadenspannung F_0 rechnen. Außerdem ist für kleine Winkel $\sin(\alpha) \approx \tan(\alpha)$. Das ist wichtig, denn die Formel oben benutzt $(z_j - z_i)/ds = \tan(\alpha)$. Die z -Komponente einer schräg einwirkenden Zugkraft wäre aber $F_0 \sin(\alpha)$.

Vorsicht! Bei zu großer Maschenweite ds , zu großen Zeitschritten dt , zu vielen Zeitschritten insgesamt und zu großem Verhältnis dz/ds entartet die Berechnung irreparabel. Die Ergebnisse werden dann ungenau oder gar unbrauchbar und zeigen nicht oder nur sehr ungenau die gesuchte Lösung. Vergleichen Sie als Ingenieur Ihre berechneten Lösungen immer mit Messungen bzw. anderen Lösungen, zumindest aber mit Ihrer Vorstellung der Wirklichkeit.

10.2 Iterative Berechnung der Lösung

Bei der iterativen Lösung benötigen wir speziell in diesem Fall der elastischen, nicht steifen Membran keine Koeffizientenmatrix \mathbf{A} , sondern nur die Auslenkungsmatrix \mathbf{z} und die Geschwindigkeitsmatrix $\mathbf{zP}(\dot{z})$. Zu jedem inneren Knoten werden die 4 direkten Nachbarn gesucht, ihre z-Koordinaten addiert und die z-Koordinate des mittleren Knotens 4 mal subtrahiert. Mit Faktor F_0/ds erhalten wir die Resultierende $Fzdn_{ij}$ der Stringkräfte der direkten Nachbarn in z-Richtung.

$$Fzdn_{i,j} = \frac{F_0}{ds} (z_{i,j-1} + z_{i,j+1} + z_{i-1,j} + z_{i+1,j} - 4z_{i,j}) .$$

Wir verbessern das elastische Verhalten der simulierten Membran, indem wir auch Stringkräfte zu den 4 indirekten Nachbarn einführen, d.h. schräge Verbindungslinien zu den 4 Knoten im Winkel 45° , 135° , 225° und -45° . Hier müssen wir aufgrund des um Wurzel 2 größeren Abstands zwischen indirekten Nachbarn die folgende Formel für die Resultierende $Fzidn_{ij}$ verwenden:

$$Fzidn_{i,j} = \frac{F_0}{ds * \sqrt{2}} (z_{i-1,j-1} + z_{i-1,j+1} + z_{i+1,j-1} + z_{i+1,j+1} - 4z_{i,j})$$

Die Gesamtstringkraft ist dann die Summe $Fz_{ij} = Fzdn_{ij} + Fzidn_{ij}$. Mit dieser Kraft wird die Knotenmasse m beschleunigt. Die Beschleunigung ist

$$\ddot{z}_{i,j} = Fz_{i,j} / m .$$

Mit einem willkürlich festzulegenden Zeitschritt dt wird dann die neue Geschwindigkeit \dot{z} des Knotens integriert:

$$\dot{z}_{i,j}^{neu} = \dot{z}_{i,j}^{alt} + \ddot{z}_{i,j} \cdot dt .$$

Die neue Position z ergibt sich durch eine weitere Integration:

$$z_{i,j}^{neu} = z_{i,j}^{alt} + \dot{z}_{i,j} \cdot dt .$$

Ist der Knoten im Kräftegleichgewicht, dann ist die Beschleunigung Null. Wir verändern die Geschwindigkeit und die z-Koordinate z_{ij} dieses Knotens nicht. Ist $Fz_{ij} > 0$, dann sind die aufwärts gerichteten Kräfte größer. Der Knoten beschleunigt nach oben. Ist $Fz_{ij} < 0$, dann sind die abwärts gerichteten Kräfte größer. Der Knoten beschleunigt nach unten.

Die Rand- und Startbedingungen haben einen großen Einfluss auf den Verlauf der Lösung. Wir nehmen sogenannte Dirichletsche Randbedingungen, d.h. auf dem Rand ist z und \dot{z} Null. Die Startbedingung ist bei uns, dass auch alle inneren Knoten Werte z und \dot{z} gleich Null haben mit Ausnahme von einem oder zwei speziell ausgesuchten Knoten, den punktförmigen Quellen. Dort setzen wir z gleich einer zeitabhängigen Störfunktion. Im Beispiel ist das ein Kosinus mit der Eigenfrequenz eines Knotens, bei dem die 8 Nachbarn fixiert sind:

$$\omega = \sqrt{\frac{c}{m}} \quad \text{mit der Federkonstanten} \quad c = 4 \left[\frac{F_0}{ds} + \frac{F_0}{ds \sqrt{2}} \right] .$$

Bei der Wahl des Zeitschritts müssen wir vorsichtig agieren. Im Beispiel wird der kleine Wert $dt=0.000001$ verwendet. Durch Versuche können Sie den maximal möglichen Zeitschritt festlegen. Da wir eine Animation erstellen wollen, speichern wir immer nach einer Bildzeit das Momentanbild unserer Membran ab. MATLAB gestattet dann anschließend das Abspielen der Bilder in Form einer Videosequenz (Funktion `movie`). Die Bildzeit legen wir

so fest, dass etwa 150 Bilder gespeichert werden. Die Endzeit t_E der Simulation legen wir so fest, dass die simulierte Welle sich gerade über das Becken ausbreiten kann. Im Beispiel ist $t_E=0.3$ Sekunden, die Bildzeit 0.002 Sekunden.

Da MATLAB bei Graphiken die Skalierung der Maßstäbe den dargestellten Daten anpasst, setzen wir zwei Randknoten auf z-Werte, die von den anderen Knoten nie erreicht werden, aber auch nicht so groß sind, dass unsere erzeugte Welle zu flach erscheint.

Randknoten sind fixierte Knoten. Ihre z- und \dot{z} -Werte werden nicht verändert. Dabei hilft uns eine Markierungsmatrix **RKn** von der exakt gleichen Größe wie unser Maschennetz. Für jeden Randknoten legen wir mit $RKn = \mathbf{1}$ fest, dass dieser Knoten tabu ist. Seine Koordinaten sind fixiert. Bei den inneren Knoten ist $RKn = \mathbf{0}$. Diese Knoten dürfen sich in z-Richtung bewegen.

Stellen Sie sich ein Becken vor mit Wasser gefüllt, in das in konstanten Zeitabständen immer auf dieselbe Stelle Steine geworfen werden (Punktquelle). Das Becken ist in x-Richtung $N_x \cdot ds$ Meter lang, in y-Richtung $N_y \cdot ds$ Meter breit. Die kleine Größe **ds** ist die Maschenweite unseres Gitternetzes, mit dem wir die Wasseroberfläche simulieren.

Beispielprogramm für eine iterative Lösung über der Zeit:

<pre>ds=0.03; Fo=100; Nx=52; Ny=52; dt=0.000001; M=1; Bildzeit=0.002;</pre>	Festlegung der Konstanten.
<pre>Fods=Fo/ds; Fodsw2=Fo/(ds*sqrt(2.0)); omega=sqrt(4*(Fods+Fodsw2)/M);</pre>	Häufig benötigte Zwischenwerte
<pre>[x, y]=meshgrid(ds:ds:(Nx*ds) , ds:ds:(Ny*ds));</pre>	Herstellung der x-y-z-Koordinaten-Matrizen für die spätere Graphik.
<pre>RKn=zeros(Ny,Nx);</pre>	Randknotenmatrix RKn mit Nullen belegen.
<pre>RKn(:, 1)=1; RKn(:, Nx)=1; RKn(1, :)=1; RKn(Ny, :)=1; RKn(:, 2)=1; RKn(:, Nx-1)=1; RKn(2, :)=1; RKn(Ny-1, :)=1;</pre>	Linke und rechte Knotenreihe. Obere und untere Knotenreihe. Dasselbe für die zweiten, weiter innen liegenden Reihen.
<pre>RKn(3 : 25, 34 : 36)=1;</pre>	3-fache Reihe von Randknoten für den Steg markieren.
<pre>RKn(36, 22)=1;</pre>	Jede Quelle wird wie ein Randknoten behandelt.
<pre>z=zeros(Ny,Nx); zP=zeros(Ny,Nx);</pre>	zMatrix Herstellen und Null setzen. z-Punkt-Matrix.
<pre>z(:, 1)=0.001; z(:, Nx)=0.001; z(1, :)=0.001; z(Ny, :)=0.001;</pre>	Die äußeren Randknotenreihen stellen den erhabenen Beckenrand dar mit Höhe $z=0.001$.
<pre>z(3 : 24, 35)=0.001;</pre>	Der Steg ebenfalls erhaben.
<pre>z(1,1)= -0.01;</pre>	Fixierter maximaler negativer z-Wert.

<pre> z(Ny,Nx)= 0.01; Fz=0; Fzdn=0; Fzidn=0; zPP=0; t=0; tE=0.3; tBild=0; nBild=0; while t<=tE; z(36, 22)= -0.01*cos(omega*t); for j=3:Nx-2; for i=3:Ny-2; if RKn(i,j)==0; Fzdn=0; Fzidn=0; Fzdn=Fzdn+z(i,j-1); Fzdn=Fzdn+z(i,j+1); Fzdn=Fzdn+z(i+1,j); Fzdn=Fzdn+z(i-1,j); Fzidn=Fzidn+z(i-1,j-1); Fzidn=Fzidn+z(i-1,j+1); Fzidn=Fzidn+z(i+1,j-1); Fzidn=Fzidn+z(i+1,j+1); Fz=(Fzdn-4*z(i,j))*Fods; Fz=Fz+(Fzidn-4*z(i,j))*Fodsw2; zPP=Fz/M; zP(i, j)=zP(i, j)+zPP*dt; z(i, j)=z(i, j)+zP(i, j)*dt; end; end; end; t=t+dt; tBild=tBild+dt; if tBild>=Bildzeit; nBild=nBild+1; surf(x,y,z); </pre>	<p>Fixierter maximaler positiver z-Wert.</p> <p>Deklaration von Variablen, die in den Schleifen benötigt werden.</p> <p>Startzeit.</p> <p>Endezeit.</p> <p>Start einer Bildzeit.</p> <p>Bildzähler.</p> <p>Start der Zeitschleife von t bis t_E.</p> <p>Störfunktion als Punktquelle.</p> <p>In einem Zeitschritt werden nacheinander alle inneren Knoten behandelt. Die Knotenindizes i und j laufen von j=3,4,...,Nx-2, i=3,4,..., Ny-2.</p> <p>Wir nehmen nur freie Knoten.</p> <p>Startwerte der beiden Kräftesummen.</p> <p>Die 4 direkten Nachbarn: Beitrag linker Nachbar. Beitrag rechter Nachbar. Beitrag Nachbar darunter. Beitrag Nachbar darüber.</p> <p>Die 4 indirekten Nachbarn: Beitrag linker unterer Nachbar. Beitrag rechter unterer Nachbar. Beitrag linker oberer Nachbar. Beitrag rechter oberer Nachbar.</p> <p>Umrechnung der z-Differenzen in Krafteinheiten Newton.</p> <p>Beschleunigung des Knotens. Neue Geschwindigkeit des Knotens. Neue z-Koordinate des Knotens.</p> <p>Ende der if-Anweisung. Ende der i-Schleife. Ende der j-Schleife.</p> <p>Zeiteinheiten zählen. Bildzeit zählen.</p> <p>Ist eine Bildzeit um? Bildzähler erhöhen. Graphik des Momentanzustands.</p>
--	---

<pre>Bildspeicher(nBild) = getframe; tBild=0; end; end; movie(Bildspeicher)</pre>	<p>Graphik abspeichern. Bildzeit neu starten fürs nächste Bild. Ende der if-Anweisung.</p> <p>Ende der Zeitschleife.</p> <p>Abspielen der gespeicherten Bilder.</p>
---	---

